International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 11 (2018) pp. 10005-10010
© Research India Publications. http://www.ripublication.com

Detecting Software Bad Smells from Software Design Patterns using Machine Learning Algorithms

Akashdeep Kaur[1] **and Satwinder Singh**[2]

[1]M. Tech Student, [2]Assistant Professor
Central University of Punjab, Bathinda, Punjab-151001, India.

Abstract

Design patterns and smells are two approaches that assure design quality. Code smells are symptoms of the design issues which can further hinder the maintenance of the software system. Software Design Patterns represent solutions to various design problems. The code having good design patterns is expected to have less bad smells but their inadequate use can lead to more design smells. Therefore, a code having design patterns can have design smells. This study finding some design pattern and smell pairs which co-exist in the code. This study uses J48 supervised machine learning algorithm to detect the desired relation. The model presented in this paper successfully finds the desired results.

Keywords: Design Patterns, Design Smells, Software Maintenance, Software Quality

INTRODUCTION

Software Design Process in software engineering deals with the activities that are to be performed for the successful completion of a software [1]. The Software Design Process includes activities such as requirement analysis, designing, coding, testing and maintenance. Software Maintenance is the most important, difficult and the costlier phase of software development. Design Patterns are a way to make the software maintenance easier. Software design smells are the structures that indicate violation of the fundamental design principles and their negative impact on design which deteriorate design quality. Cost of repairing the defects that occur at the beginning of the software development cycle is lower.

Design patterns represent recommended solutions to design problems and the design smells are indication of the presence of design problems. These design smells can hinder the maintenance process of the software. We can say that both design patterns and design smells are correlated to each other as the presence of one is the absence of another. If there are design smells in the software, then design smells will be absent and vice versa.

In other work the relationship between design patterns and bad smells has been found that whether the classes participating in the design patterns have fewer bad smells than the classes that do not participate in design patterns or not. The results showed that the classes that participate in design patterns have fewer bad smells than the classes that do not participate in design patterns. Taking that result into account, this study focuses on finding the patterns-smell pairs that co-exist and to find those patterns that do not contain any smell in them.

In this research we run software pattern detection and smell detection tools to extract the patterns and smells from various systems. The results of this study shows some design patterns that do not contain any smells and also some pattern-smell pairs that co-exist.

This section introduces this study. The remainder of this paper is organized as follows. Section 2 presents the background on the concepts of this study. Section 3 describes Research methodology. Section 4 is about the techniques used to accomplish this study Section 5 presents the procedure to complete this study. Section 6 presents the results obtained. Section 7 presents the conclusion and future work.

BACKGROUND

Design Patterns: Software design Pattern is a general reusable solution to the problems that commonly occur during the software development. Software design patterns shows the relationship between classes and objects. Four authors namely Erich Gama, Richard Helm, Ralph Johnson and John Vlissides in 1994 published a book, Gamma et al. [1] on Design Patterns-Elements of Reusable Object-Oriented Software, which was beginning to the concept of software design patterns. The authors collectively called themselves Gang of Four(GOF). According to GOF the design patterns are based on two principles:

1. Program to an Interface, not Implementation.
2. Favor object composition over Inheritance.

Abstract Factory pattern provides an interface that allows creation calls to one or more concrete classes so as to deliver specific objects. Singleton pattern allows that only one instance of class is allowed within a system. It is used where exactly one instance of an object is needed Adapter pattern allows the classes with disparate interfaces to work together by creating common object with which they may interact and communicate. Bridge pattern defines abstract object structure independent of the implementation object structure to limit coupling Proxy Pattern acts as a pass through entity or placeholder object allows the object level access control. Template Method identifies the framework of an algorithm thus allows implementing classes to define the actual behaviour.

Design Pattern Detection Tool: Detection of design pattern can be useful in many ways for e.g., to gain good comprehension of a software system and of the problems addressed during system development. The design patterns applied on a software system form a dictionary of the design solution, thus simplifying the communication between the developer and the maintainer. As discussed earlier the presence of design patterns indicate good software health thus making them reusable.

Web Of Patterns: Web of Patterns is developed by Dietrich and Elgar. The development is based on the OWL ontology which is published on the projects web page. This technique uses the formula described using the first order logic

Bad Smells: Long Method is too long which, makes it difficult to understand, extend and change. The longer the method the more it is difficult to understand it. All the time you have to use extract method which means you have to shorten the method. If you use extract method then you may end up passing lot of perimeters, then you can use Replace Temp with query to eliminate temps and shorten the long list of the perimeters passed. Large C lass bad smell means that the class is trying to do so much work. these classes may have too many methods or instance variables, may have duplicated code. The classes which do too much work may cause duplication. Extract subclass and extract class can be used in case of this smell. long parameter list smell occurs when the parameter list is too large making it difficult to understand. Long parameter list of needed when you every time need some new data for the method to act upon. Using the Replace parameter with Method you can have data in one parameter by making a request of object that you already know. Refused Bequest smell occurs when the derived class do not use all the methods or the data it inherits from its parent class. Its bad case occurs when a class refuses to implement an interface. In Lazy Class each created class needs money and time to understand and maintain. A lazy class is a class which does nothing or nothing useful and this should be removed. If there is a subclass that is not doing enough then you can collapse that hierarchal and classes which are nearly useless should be converted to inline class. Data Class is a class containing data without any logic, but a should contain both data and logic. These classes are dumb classes and can be manipulated by other classes

Bad Smell Detection Tool: Software inspection is a well-known technique for improving the quality of code which involves careful examination of the design, code and documentation of the system to check for the potential problems based on the past experience [2]. Code smells can be automatically detected using automated code smell detection tools and the smells can be visualized in the code. There ae many automation bad smell detection tools available each of which id capable to extract some bad smell from the code of different type. iPlasma**:** this tool is integrated platform for quality assessment of object oriented systems which include support for all the necessary phases of analysis, model extraction, high level, class based analysis. iPlasma is able to detect identity disharmonies, code disharmonies, collaboration disharmonies and classification disharmonies. Some code smells are also considered as disharmonies such as duplicate code, God class, Refused parent Bequest.

LITERATURE REVIEW

Cardoso and Figueiredo [13] report the importance of design patterns and consider that these solutions are considered to be good programming practices and are solution reusable solutions to a re-occurring problems in software design. Bad smells are considered to be the symptoms of something wrong in the code or system design. An exploratory study is performed in order to identify the relationship and co-occurrences of design patterns and bad smells. A study on five systems is conducted to find co-occurrences of design patterns and bad smells. Before this study Cardoso and Figueiredo conducted one more study which performed literature review in order to understand the bad smells and design patterns together, Cardoso and Figueiredo proposed some tools to refactor the code fragments to refactor the patterns. In order to extract the co-occurrences of design patterns and code smells they rely on association rules to indicate how strong relationship the design patterns and code smells have. For example, association rules indicate how often the factory method and feature envy are present in a class at the same time. **Walter and Alkhaeir [14]** carried out an exploratory study with the goal to 1. Determine if and how the presence of design patterns is related to the presence of bad smells, 2. Investigate if and how the presence of these relations changes along with the code and 3. Identify relationships between design patterns and design smells. To achieve these objectives, the authors performed analysis of the evolution of the two systems: Apache Maven and JfreeChart. Total 87 versions were analyzed in which 37 were related to the first system and 55 to the second system. In addition to this the study was conducted with 7 different types of bad smells and nine design patterns. The authors used non-parametric trend Mann Kendall test and association rules, then the authors performed the analysis of the information extracted. The result of their study shown that the presence of design patterns is related to the absence of the bad smells in same class. In simple words a class with design patterns tends to have less or no bad smells. In case of design patterns state strategy, adapter command, factory method and singleton lack bad smells. But the composite design pattern showed stronger relationship with the presence of bad smells.

PROCEDURE

To get the results a procedure was followed. The first phase in the whole procedure was to select some open software systems, then to select tools for extracting the smells and patterns from the selected source code. Then these tools are used to extract smells and design patterns. This step further involves the tuning of the classes extracting by smell detection tool and of the classes extracted by the pattern detection tool to form dataset. Second phase of this research involves choosing some machine learning algorithm in our case J48 decision tree is chosen as the classifier to get results. In next step each prepared data set is run using the machine learning algorithm. And the results are noted down in the form of tables. These tables can further be used to form graphs and charts for the better evaluation of results through which results of the smell-pattern pair that co-exist and patterns which do not have any smells will be identified.

RESULTS

This section discusses about the data collected and experiments performed on that data along with the results achieved.

To build a model using supervised learning technique to find smell and design pattern pair displaying significant relationship: To identify a smell-pattern pair such that the code having that pattern should have less smell.

Classifiers results and Data set: The following results shows the results extracted from the eclipse 3.7 version using J48 classifier. Five parameters are used to measure the performance of the proposed model: recall, precision F-Measure, PRC and ROC. Precision is called as positive predictive value. It is the fraction of the relevant instances. Recall is also known as sensitivity. It is the fraction of the relevant instances which have been retrieved in the total amount of the relevant instances. F-Measure is the measure of the accuracy of the test. It is defined as the weighted harmonic mean of precision and recall of test. ROC curve is a fundamental tool for diagnostic test evaluation the area under ROC curve is a measure of how well a parameter distinguishes between two groups.

Table 1. Performance measure of Adapter pattern having smells using for eclipse 3.7

Pattern	Smell	ROC	PRC
Adapter	Brain Class	0.462	0.846
	Data Class	0.463	0.838
	God Class	0.488	0.527
	Request parent bequest	----	----
	Schizophrenic Class	0.419	0.981

Table 1 shows the results of J48 algorithm on the data set where the classes obtained from Eclipse version 3.7 have Adapter pattern and have five type of smells in them i.e., God class, Data Class, Brain Class, schizophrenic class and God class. This dataset is created to show which smells are present in the software in the presence of adapter pattern. From the ROC and PRC values of various smells we can conclude from the calculated results that the Request Parent Bequest is absent and God Class has least presence while there are design patterns in the software.

Table 2. Performance measure of Adapter pattern having smells using J48 for eclipse 3.6

Pattern	Smell	ROC	PRC
Adapter	Brain Class	0.736	0.938
	Data Class	0.496	0.325
	God Class	0.494	0.560
	Request parent bequest	----	----
	Schizophrenic Class	----	----

Table 2 shows the results of J48 algorithm on the data set where the classes obtained from Eclipse version 3.6 have Adapter pattern and have five type of smells in them. ROC values of smells in the presence of Adapter pattern depicts that the Request Parent Bequest and schizophrenic class bad smells are absent in Eclipse 3.6 whereas data class has least presence while there are design patterns in the software.

Table 3. Performance measure of Bridge pattern having smells using J48 for eclipse 3.7

Pattern	Smell	ROC	PRC
Bridge	Brain Class	0.269	0.751
	Data Class	0.128	0.854
	God Class	0.485	0.517
	Request parent bequest	----	----
	Schizophrenic Class	----	----

Table 3 shows the results of J48 algorithm on the data set where the classes obtained from Eclipse version 3.7 have Bridge pattern and have five type of smells in them i.e., God class, Data Class, Brain Class and Request parent bequest. we can conclude from the calculated results that the Request Parent Bequest and Schizophrenic class smells are absent and God class has least presence while there are design patterns in the software. So the pair we obtain from this is Bridge-God class

Table 4. Performance measure of Bridge pattern having smells using J48 for eclipse 3.6

Pattern	Smell	ROC	PRC
Bridge	Brain Class	0.996	0.994
	Data Class	0.479	0.054
	God Class	0.432	0.500
	Request parent bequest	----	----
	Schizophrenic Class	----	----

Table 4 shows the results of J48 forest algorithm on the data set where the classes obtained from Eclipse version 3.7. The Request Parent Bequest and Schizophrenic class is not present in bridge pattern but the Data Class has minimum presence. Therefore, the pair we obtain here id Bridge- Data Class. The Data class is present in less number in bridge pattern.

Table 5. Performance measure of Template pattern having smells using J48 for eclipse 3.7

Pattern	Smell	ROC	PRC
Template	Brain Class	0.464	0.841
	Data Class	0.446	0.930
	God Class	0.489	0.546
	Request parent bequest	----	----
	Schizophrenic Class	0.149	0.994

Table 5 shows the results of J48 algorithm on the data set where the classes obtained from Eclipse version 3.7 have Bridge pattern and have five type of smells in them. From the results obtained we conclude that the request parent Bequest is not present in the presence of template pattern. But God class has least presence, so the pair obtained here is Template pattern and God class as it is present in small amount in the presence of template pattern.

Table 6. Performance measure of Template pattern having smells using J48 for eclipse 3.6

Pattern	Smell	ROC	PRC
Template	Brain Class	0.492	0.902
	Data Class	0.441	0.982
	God Class	0.696	0.676
	Request parent bequest	0.099	0.995
	Schizophrenic Class	0.829	0.992

Table 6 shows the results of J48 algorithm on the data set where the classes obtained from Eclipse version 3.6 have template pattern and have five type of smells in them. We conclude that the Request Parent Bequest is present. The God class is present in less value as compared to other smells in the code

Table 7. Performance measure of smells in Template pattern using for eclipse 3.6 and eclipse 3.7

Pattern	Smell	ROC	PRC
Template	Brain Class	0.494	0.897
	Data Class	0.453	0.972
	God Class	0.664	0.650
	Request parent bequest	0.100	0.996
	Schizophrenic Class	0.718	0.991

Table 7 shows the presence of bad smells in the presence of Template design patterns. The data taken includes the collective data of both the versions of the Eclipse software. Data is collected so as to get accurate results instead of applying algorithms on them differently. J48 algorithm is used to carry out the above model.

Table 8. Performance measure of smells in Bridge pattern using J48 for eclipse 3.6 and 3.7

Pattern	Smell	ROC	PRC
Bridge	Brain Class	0.989	0.982
	Data Class	0.288	0.888
	God Class	0.461	0.509
	Request parent bequest	----	----
	Schizophrenic Class	----	----

Table 8 shows the performance measure of J48 algorithm on the collective data of both the versions of Eclipse. From the above table we can conclude that is inherit, request parent Bequest and Schizophrenic class bad smells do not occur in the presence of bridge pattern.

Table 9. Performance measure of smells in Adapter pattern using J48 eclipse 3.6 and 3.7

Pattern	Smell	ROC	PRC
Adapter	Brain Class	0.609	0.888
	Data Class	0.487	0.847
	God Class	0.495	0.546
	Request parent bequest	----	----
	Schizophrenic Class	0.419	0.985

Table 9 shows the application of J48 algorithm on both the versions of Eclipse. From the above measures we can conclude that only Request Parent Bequest is the only smell that do not occur in the presence of Adapter pattern.

Table 10. Performance measure of smells in Singleton pattern using J48 for Eclipse 3.6 and Eclipse 3.7

pattern	Smell	ROC	PRC
	Brain Class	-	-
	Data Class	-	-
Singleton	God Class	-	-
	Request parent bequest	-	-
	Schizophrenic Class	-	-

The above table shows that the singleton pattern does not contain any type of smell.

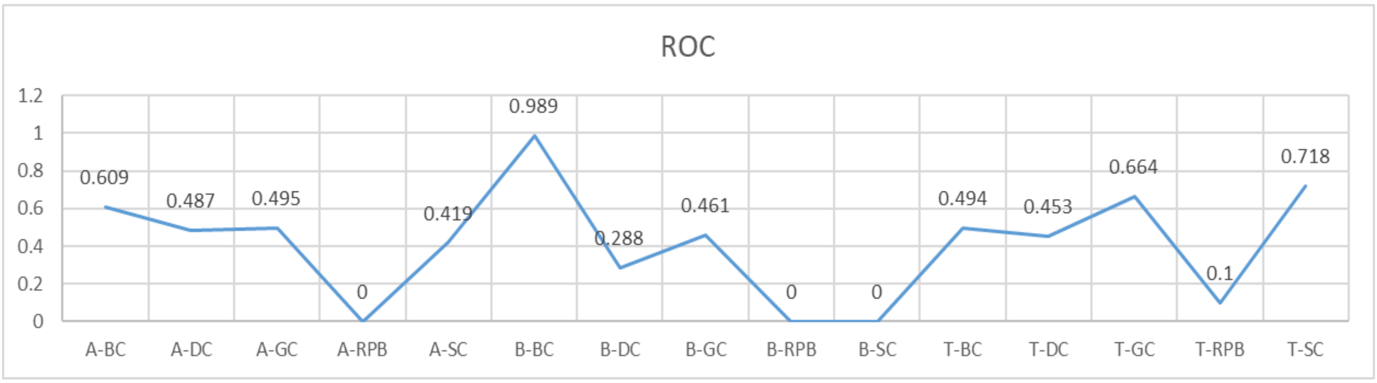

Figure 1. ROC Curve values for all the Bad smells present in each pattern.

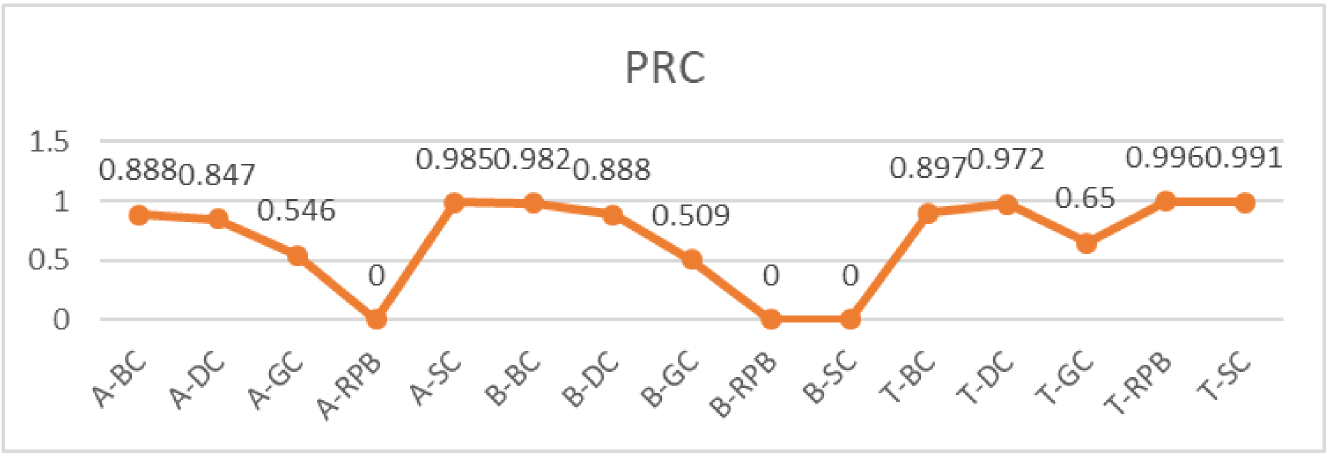

Figure 2. PRC values for all the Bad smells present in each pattern.

Figure.1 and 2 shows the ROC and PRC values for both the versions of Eclipse. The extracted data from two versions of Eclipse software of software design patterns and software bad smells and applying machine learning algorithms have given the values to find a pair, which shows the least presence of some smell in some pattern. The results of classifiers as shown in Fig 11 shows that Request Parent Bequest smell is not present in any pattern but template pattern. The Template Pattern and God Class smell, Bridge pattern and God Class smell and Adapter and God Class are pairs showing relationship.

CONCLUSION

This research work focuses on finding the relationship between the Software Design Patterns and code smells which are also known as bad smells. For this research different tools are studied for extraction of software design patterns and bad smells. The tool selected for Design Pattern extraction is Web of Patterns which is an Eclipse plugin. The tool selected for code smell detection is iPlasma. It is an integrated platform. It is capable of detecting various disharmonies in the code. The research includes use of machine learning algorithms to achieve the objectives of the research first objective was to identify those patterns which do not have any smell present in them and the second objective was to obtain some pattern smell pairs which represents the objective first that the patterns have less bad smells. The objectives are achieved by studying the ROC and PRC values obtained by the used algorithms. The results are validated using two versions of Eclipse. With the results obtained it is concluded that the Singleton Pattern shows no presence of bad smells and the Request Parent Bequest is the bad smell which shows no presence in the presence of design patterns. The pairs having relationship of least smell in the presence of patterns are Template Pattern and God Class smell, Bridge pattern and God Class smell and Adapter and God Class are pairs showing relationship.

The work can be extended by using more number of software's and using design patterns and bad smells from more number of detection tools.

REFERENCES

[1] Gamma, E., Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented software," Addison-Wesley, 1995

[2] Emden, E. V., Moonen, L(2002), "Java Quality Assurance by Detecting Code Smells", *9th Working Conference on Reverse Engineering (WCRE)*, pp. 97-106, IEEE, 2002 vm

[3] A K Dwivedi, A Tirkey, R B Ray, S K Rath (2015), “Software Design Pattern Recognition using Machine Learning Techniques”, *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–6.

[4] Antoniol, G., Fiutem, R., and Cristoforetti, L. (Nov 1998). Using Metrics to Identify Design Patterns in Object-Oriented Software. *In Proceedings of the Fifth International Symposium on Software Metrics (METRICS98),* pages 23–34. IEEE Computer Society.

[5] N. Maneerat, P. Muenchasiri (2011). *Bad Smell prediction software design model using machine learning technique.*

[6] Francesa Arcelli Fontana, Marco Zanoni (2017) *Code smell severity Classification using Machine Learning Technique.*

[7] Francesa Arcelli Fontana, Marco Zanoni, Mika V Mantyla, Alessandro Marino. (2015) *Comparing and experimenting machine learning techniques for code smell detection.* New York.

[8] Francesca Arcelli Fontana, Elia Mariani, Andrea Morniroli, Raul Sormani, Alberto Tonello(2011) *an experience report on using code smells detection too* Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011

[9] Bruno L. Sousa, Mariza A. S. Bigonha, Kecia A. M. Ferreira, Finding smells: flexible composition of bad smell detection strategies. 2017 IEEE 25th International Conference on Program Comprehension(ICPC)

[10] Iqbal Muhammad and Zhu Yan, Supervised Machine learning approaches: a survey ictact journal on soft computing, april 2015, volume: 05, issue: 03

[11] Wanwangying Ma, Lin Chen, Yuming Zhou, Baowen Xu, Xiaoyu Zhou. (2014) Are Anti-Patterns Coupled? An Empirical Study Conference: 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS).

[12] Tina R. Patil, Mrs. S. S. Sherekar, Performance Analysis of Naive Bayes and J48 Classification Algorithm for Data Classification. International Journal of Computer Science and Applications Vol. 6, No.2, Apr 2013

[13] Cardoso, B. and Figueiredo, E (2015). Co-occurrences of design patterns and bad smells in software systems: An exploratory study. Annual Conference on Brazilian Symposium on Information Systems. Vol 1 page 46, Goias, Brazil.

[14] Walter, B., Alkhaeir, T (2016) The relationship between design patterns and code smells: An Exploratory study.

[15] Satwinder singh, Sharanpreet kaur, Systematic Literature Review: Refactoring For Disclosing Code Smells In Object Oriented Software, “Ain Shams Engineering Journal” ,22 March 2017.

[16] S Singh, KS Kahlon ,”Effectiveness of refactoring metrics model to identify smelly and error prone classes in open source software “,ACM SIGSOFT Software Engineering Notes 37 (2), 1-11

[17] Satwinder Singh, Sharanpreet kaur, “Prediction Model to Investigate Influence of Code Smells on Metrics in Apache Tomcat “, International Journal of Advanced Research in Computer Science Volume 8, No. 5, May – June 2017.