SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 9(1); June (2022), pp. 43–48
SSN (Online): 2352-6386; ISSN (Print): 2405-9021
https://alife-robotics.org/jrnal.html

Research Article

Detection of Blob and Feature Envy Smells in a Class Diagram using Class's Features

Bayu Priyambadha[1], Tetsuro Katayama[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]
[1]*University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192, Japan*
[2]*Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, 851-2195, Japan*

ARTICLE INFO

Article History

Received 20 November 2020
Accepted 10 August 2021

Keywords

Smell Detection
Class Diagram Smell
Design Quality
Software Design

ABSTRACT

Measuring the quality of software design artifacts is difficult due to the limitation of information in the design phase. The class diagram is one of the design artifacts produced during the design phase. The syntactic and semantic information in the class is essential to consider in the measurement process. Smell detection uses class-related information to detect the smell as an indicator of a lack of quality. Several classifiers use all information related to the class to prove how informative it for the smell detection process. The smell types that are a concern in this research are Blob and Feature Envy. The experiment using three classifiers (j48, Multi-Layer Perceptron, and Naïve Bayes) confirms that Blob smell detection utilizes the information successfully. On the other hand, Feature Envy still needs more elaboration. The average true positive rate by the classifiers is about 80.67%.

1. Introduction

Managing the quality of the software is begun from the earliest phase of software development. Therefore, qualified software artifact resulting from every phase of development is essential to maintain. The excellent quality of the software artifact affects the final result of the software product. Therefore, it is necessary to measure the quality of the software artifact to preserve the quality of software artifacts. The measurement of software artifacts makes a judgment about the best result of software product[1] and the process improvement in software development[2]. Often, the approach to measuring the quality of software artifacts uses the metric quality. Therefore, the measurement process results in the value of the software quality level.

Many matrices can be used as a tool to measure the quality of software artifacts. For example, is matrices that related to the object-oriented approach in software development. In an object-oriented approach, the quality measurement is often done to the quality indicators of the software artifacts, for example, complexity, cohesion, and coupling[3]. Those three indicators are used to the class as a fundamental object in the object-oriented approach. Class is the smallest unit in the object-oriented system. The software consists of many classes as the template or blueprint of the objects. Every object represents the thing that exists inner the software. The object has specific characteristics and abilities. And, the object also can

Corresponding author's E-mail: bayu@earth.cs.miyazaki-u.ac.jp, kat@cs.miyazaki-u.ac.jp, kita@sun.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@cs.miyazaki-u.ac.jp, oka@cs.miyazaki-u.ac.jp

work together with other entities to accomplish the particular functionality of the software. Therefore, a good design of the class as the representation of the object can impact the overall software.

There are two points of view in assessing the class quality, from the inner and outer of the class. First, cohesion means the measurement from the inner of class means measuring relationships between internal elements. Second, coupling means the measurement of the class is based on the relationship of one class with another class. Both indicators are essential to inform us about the compactness of the class itself. External quality attributes are related to the compactness of the class. There are two examples of external quality attributes, maintainability, and understandability. The more level of compactness of the class, the class is easier to change and understand. To change and understand the class, one only has to consider the class itself.

Nowadays, several researchers are extensively discussing and evaluating software design problems, specifically in an object-oriented approach. The thing related to the design problem is the smell. The other term is "code smell" [4], or "design flaw" [5]. Generally, the existence of smell indicates something misplaced in the software artifact that can affect the quality attributes of the software.

Both the metric and the existence of smells indicate the quality of software artifacts. The metric can be used as a tool to measure the quality related to the quality attribute. And, the existence of smell indicates there is a lack of quality. The relationship between software metrics and the presence of smells is an exciting thing to learn. Bigonha et al. explain the usefulness of the software metric threshold for detecting bad smells and fault prediction[6]. This research expressed that there is a relation between the metric and the presence of smell. The use of metrics to detect smell is very effective by using specific thresholds. Several bad smells have been proposed to detect, for example, Large Class, Long Method, Data Class, Feature Envy, and Refused Bequest based on the metrics. And, it also uses several matrices to detect the smell, for example, DIT (Depth in Inheritance Tree), LCOM (Lack of Cohesion between Methods), NOF (Number of Fields), NOM (Number of Methods), NORM (Number of Overridden Methods), NSC (Number of Children), NSF (Number of Static Attributes), NSM (Number of Static Methods), SIX (Specialization Index) and WMC (Weighted Methods per Class). All matrices are the class level's metric. All metrics are measured on the source code of the software products. So, this research work as the result of the implementation phase of software development.

Smell detection is essential to know where the lack is present in the part of the software. The position of smell becomes a direction for the developer to make improvements to the software artifact. The refactoring activity can overcome the smell. Fowler defines the term refactoring as changing internal structure that is not changing the external behavior of software[4]. Along with the software evolution, the code is modified over time, and the structure of the code will gradually disappear. The bad smell is emerging, and the code quality is decaying as stated in Lehman's Laws that said as the software evolves, the structure is degraded[1]. Refactoring is the effort that we have to pay to manage the structure of code.

Mostly smell detection and refactoring process is done at the level of source code. The researcher is concerned about the importance of shifting the work to the design phase based on design artifacts. Start from the smell detection in design[7], then continue to the refactoring in design[8,9]. This effort is aimed at maintaining the quality of the artifacts as early as possible. The challenge to do the smell detection on the design artifact is the limitation of information. Collecting the useable data to detect the smell as much as we can on source code is hard. At the design level, there is no source code. There is only the model of the software that it wants to build. One of the models as the product of the design is the class diagram. The class diagram informs about the structure of the software based on the class and their relationship. The possibility to collect the information from the class diagram is by understanding the syntactic and semantic information that may exist[10].

This research aims to collect the information based on the syntax and semantic information that exists in the Class Diagram. Then use the information to detect the existence of the smell at the design artifact. This research uses the inner information of class (related to cohesion) to detect the smell. Two types of smell that are considered related to cohesion are Blob and Feature Envy, as stated by Bigonha et al. that the LCOM metric is related to the Feature Envy smell[6]. There are seven of information that will be extracted from the class diagram,

number of attributes, number of methods, number of the relation between method and attribute, number relation between method and method, number relation between attribute and attribute, the capacity of relation inner the class and cohesion value. The Landfill dataset will complement the seven information features from the class to generate the dataset[11]. That information will be classified using three classifiers, J48, Multi-Layer Perceptron (MLP), and Naïve Bayes, and compare the result. This research focuses on how the usefulness of the information can be used to detect the smell.

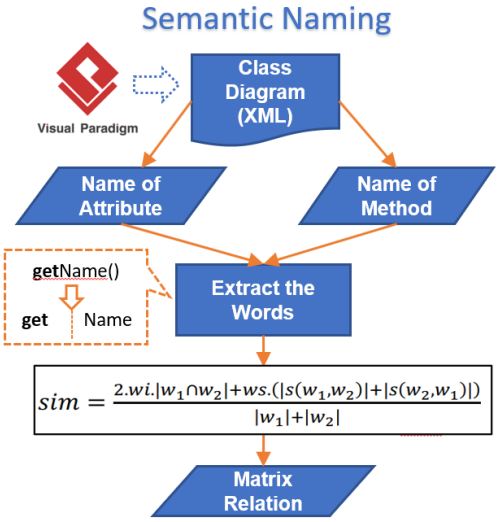

Fig. 1. Process of Semantic Similarity Analysis of the Label Name

The rest of the paper is organized as follows. In section 2, we present every data used in this research and how to get those data. Section 3 describes how to label every piece of data. Section 4 describes the whole process of classification. Section 5 describes the result and discussion. Then the last is the conclusion and future work in section 6.

2. Seven Information

There is seven information as the candidate data to use in smell detection. The data is related to one of the internal quality attributes called cohesion. The data are number of attributes, number of methods, number of the relation between method and attribute, number relation between method and method, number relation between attribute and attribute, the capacity of relation inner the class, and cohesion. The technique to get those data is interesting to discuss.

It is essential to consider the syntactic (structural) and semantic information in collecting the data. Structural information is the information that we can directly read or extract from the class diagram. And semantic information is the information that needs a little processing (interpret the meaning) to extract. In this research, the object of study is the Class Diagram that resulted from using the Visual Paradigm tools. Then, it will convert to the XML file then called XML-based Class Diagram (Fig.1).

2.1. *Number of Attributes*

Two things may exist inner the class. The first is attributes. The attributes are the field used to save the data that tells about the characteristics of the objects. Finding the attributes in the class diagram is easy. The notation for the attribute's name and the type of attribute is clearly described in the class diagram. The information about attributes is extracted from XML based class diagram.

2.2. *Number of Methods*

Methods describe the behavior or ability that the object can perform to accomplish the functionality. The name, return value type, and the method's parameters are also clearly described in the class diagram. Collecting all information about the method is included in the syntaxis or structural information.

2.3. *Number of Relation between Method Attribute*

The relation between Method and Attribute (MAR) can be counted by considering syntactic and semantic information. The relation between method and attribute is determined based on the similarity of type (syntaxis) and the similarity of meaning (semantic) between method and attribute. Fig.1 describes the process to measure the closes meaning between the label name of method and attribute. First, all label name is extracted from the XML's based class diagram. Then separate the words in the label of the name. After that, by using the semantic similarity formula by Dijkman, the similarity of meaning between the set of words is calculated[12]. If the label name between attribute and method is semantically similar, it will be considered a relation. Matrix relation is created to make it easier.

2.4 *Number of Relation between Method-Method and Attribute-Attribute*

Both the type of relation is called transitive relation or indirect relation. That relation is determined based on the direct relation between attribute and method. There are two definitions to determine method-method (MMR) and attribute-attribute relation (AAR). There will be a relation between methods if two methods are related to the same attribute. Then, there will be a relation between attributes if two attributes are associated with the same method. This type of relation is included in the category of semantic information. Fig. 2 shows the analogy of both relations.

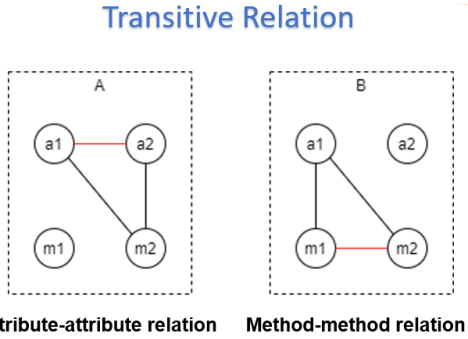

Fig. 2. The Description of Transitive Relation to Define the Relation

2.5 *Relation Capacity*

The capacity of the network inspires the calculation of the capacity or maximum relation. The maximum relation is the maximum number of relations that possibly exist in one area. Here, the class is assumed as the area of the relation. That's why the capacity is the maximum relation that can exist inner the class. The maximum relation is calculated by using Eq. 1.

$$MaxRelation = \frac{(m+a)((m+a)-1)}{2} \quad (1)$$

Where m is the number of methods and a is the number of attributes.

2.6. *Cohesion value*

The cohesion value calculation is inspired by the calculation of the graph's cohesiveness by using the network density approach. Because the form of a graph is similar to the network graph, then the formula is considered applicable to cohesion. The cohesion is calculated by considering method-attribute relation, method-method relation, and attribute-attribute relation. The relation is direct relation and transitive relation. The relations between method and attribute are divided by the maximum relation (1) that possibly exists in one class. The cohesion calculation is expressed as Eq. 2.

$$Cohesion = \frac{MAR+MMR+AAR}{MaxRelation} \quad (2)$$

3. Dataset

The Landfill dataset is a set of data managed by several researchers from several countries. It consists of 243 instances of five types of code smells identified from 20 open-source software projects[11]. In the Landfill dataset, all classes containing bad smells are labeled based on the type of smell.

Table 1. Representation of Dataset

No.	Fields	Type
1.	MAR	Numeric
2.	MMR	Numeric
3.	AAR	Numeric
4.	Number of Attributes	Numeric
5.	Number of Methods	Numeric
6.	Capacity	Numeric
7.	Cohesion	Numeric
8.	Label	Blob, Feature Envy, No Smell

This research dataset used to experiment is composed of the seven information and the label of smell provided by the Landfill dataset. The representation of the dataset is described in Table 1. The main dataset is extracted from six projects. The projects are jEdit, jHotDraw, FreeMind, HSQLDB, aTunes, and ArgoUML. The dataset consists of 300 data based on the smell class listed by the Landfill dataset and the smell-free classes, then used as training data, and also generated 49 data used for testing data.

4. Classification

In this experiment, the process of detecting the smell will use the classification method. The seven information collected using the mechanism described in section 2 will be used as the dataset. In addition, the dataset will combine with the Landfill dataset to get the class label of data as described in section 3. This research focuses on how the usefulness of the information can be used to detect the smell. Three classifiers were used in this research, j48, Multi-Layer Perceptron, and Naive Bayes. The tool used in this experiment is Weka as a Machine learning software to solve data mining problems[13]. Dataset for training and testing are covert to CSV file then import into Weka tools. All classifiers are run by using a basic configuration. Using a classifier is only the way to prove

how the dataset can be distinctive to the bad smell (Blob and Feature Envy).

5. Result and Discussion

This section will explain the result of the experiment by using the dataset and three classifiers. All is done by using Weka tools. The recapitulation of the result is described in Table 2.

J48 was able to classify (TPR) about 51% correctly. The correct classify for every label of data is, for Blob data is 94.7%, Feature Envy data is 25%, and No data is 22.2%. All labels of data can be recognized using this approach. MLP was able to classify 51% of the data correctly.

Table 2. Recap of The Testing Result

No.	Classifier	Blob	Feature Envy	No Smell	True Positive Rate (TPR)
1.	J48	94.7%	25.5%	22.2%	51%
2.	MLP	94.7%	0%	38.9%	51%
3.	Naïve Bayes	52.6%	16.7%	55.6%	44.9%

Based on the label data, Blob data is 94.7%, Feature Envy data is 0%, and No data is 38.9%. Therefore, the last classifier, Naive Bayes, can classify 44.9% of data correctly. Based on the label data, Blob data is 52.6%, Feature Envy data is 16.7%, and No data is 55.6%.

Based on the result (Table 2), two classifiers have a TPR value above 50%. The classifiers are j48 and MLP. But, the MLP classifier cannot identify the Feature Envy smell because the TPR for Feature Envy of MLP is 0%. The j48 and Naive Bayes can detect the Feature Envy smell even though with the low rate. All classifiers can detect Blob smell with a TPR above 50%. The average TPR for Blob smell is 80.67%. The characteristics expressed in the dataset led to the identification of Blob smell. However, the existing information is not enough used to find both types of smells. More detailed data is needed to improve the differentiation between the two types of smells.

6. Conclusion and Future Work

The use of seven information as a class dataset to identify the Blob smell in the class diagram is very effective. The TPR above 50% proves the effectiveness. On the other hand, it is not very good to use to identify the Feature Envy smell. The dataset is not express the big differential for Blob and Feature Envy. It makes the classifier hard to identify each type of smell. Only j48 and Naive Bayes can identify the Feature Envy but in the low of TPR.

To continue the research, it needs more exploration to increase the differentiation between both smells. The additional information of class maybe would be worth finding to increase the richness of data. The other, weighing every variable (MAR, MMR, AAR), is also considered essential to sharpen the dataset's differences.

References

1. Sommerville, I., *Software Engineering 9th Edition*, Harlow, England: Addison-Wesley Professional, 2010
2. Chidamber, S. R. & Kemerer, C. F. *A Metrics Suite for Object Oriented Design*. IEEE Trans. Softw. Eng. **20**, 1994 , pp. 476–493.
3. Chowdhury, I. & Zulkernine, M. *Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities*. J. Syst. Archit. **57**, 2011 , pp. 294–313.
4. Fowler, M. *Refactoring Improving the Design of Existing Code*. (Pearson Education - Wesley, 2019).
5. Riel, A. J. *Object-oriented Design Heuristics*. Addison-Wesley Publishing Company, 1996
6. Bigonha, M. A. S. *et al. The usefulness of software metric thresholds for detection of bad smells and fault prediction*. Inf. Softw. Technol. **115**, 2019 , pp. 79–92
7. Sidhu, B. K., Singh, K. & Sharma, N. *A Catalogue of Model Smells and Refactoring Operations for Object-Oriented Software*. Proc. Int. Conf. Inven. Commun. Comput. Technol. ICICCT 2018, 2018, pp. 313–319 doi:10.1109/ICICCT.2018.8473027
8. Misbhauddin, M. & Alshayeb, M. *Model-driven refactoring approaches: A comparison criteria*. Proc. - African Conf. Softw. Eng. Appl. Comput. ACSEAC 2012, 2012 , pp. 34–39 doi:10.1109/ACSEAC.2012.20
9. Dharmawan, T. & Rochimah, S. *Systematic literature review: Model refactoring*. Proc. 2017 4th Int. Conf. Comput. Appl. Inf. Process. Technol. CAIPT 2017, 2018 , pp. 1–5.
10. Priyambadha, B. *et al. The Measurement of Class Cohesion using Semantic Approach*. Proc. Int. Conf. Artif. Life Robot. **25**, 2020 , pp. 759–762.
11. Palomba, F. *et al. Landfill: An open dataset of code smells with public evaluation*. in IEEE International Working Conference on Mining Software Repositories vols 2015-Augus (2015). , pp. 482–485.
12. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R. & Mendling, J. *Similarity of business process models: Metrics and evaluation*. Inf. Syst. **36**, 2011 , pp. 498–516
13. Hall, M. *et al. The WEKA Data Mining Software : An Update*. SIGKDD Explor. **11**, 2009 , pp. 10–18.

Authors Introduction

Bayu Priyambadha

Bayu Priyambadha has received his Bachelor of Computer from the 10 November Institute of Technology Surabaya. He also has got a Master of Computer from 10 November Institute of Technology Surabaya. He is a member of the Software Engineering Research Group (SERG) in the Faculty of Computer Science, Brawijaya University, Indonesia. He is currently a doctoral student at the University of Miyazaki, Japan. His current research interest is Software Engineering, Software Design, Software Quality, and Software Maintenance.

Tetsuro Katayama

Tetsuro Katayama received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Yoshihiro Kita

Yoshihiro Kita received a Ph.D. degree in systems engineering from the University of Miyazaki, Japan, in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Hisaaki Yamaba

Hisaaki Yamaba received the B.S. and M.S. degrees in chemical engineering from the Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the Ph D. degree in systems engineering from the University of Miyazaki, Japan in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Kentaro Aburada

Kentaro Aburada received the B.S., M.S, and Ph.D. degrees in computer science and system engineering from the University of Miyazaki, Japan, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer networks and security. He is a member of IPSJ and IEICE.

Naonobu Okazaki

Naonobu Okazaki received his B.S, M.S., and Ph.D. degrees in electrical and communication engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE.