The Seven Information Features of Class for Blob and Feature Envy Smell Detection in a Class Diagram

Bayu Priyambadha*, Tetsuro Katayama*, Yoshihiro Kita †, Hisaaki Yamaba*, Kentaro Aburada*, Naonobu Okazaki*
**University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192, Japan*
† 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
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

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 important to consider in the measurement process. The class-related information is used to detect the smell as an indicator of a lack of quality. All information related to the class is used by several classifiers to prove how informative it to be used to detect the smell. 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 the information can be used to detect Blob smell, on the other hand, Feature Envy, still needs more research. The average true positive rate of each classifier is about 80.67%.

Keywords: Smell Detection, Class Diagram Smell, Design Quality, Software Design

1. Introduction

Good quality of the software artifacts affects the final result of software products. It is important to measure the level of its quality. The measurement is not only to make a judgment about the best result of software product[1] but also for the process improvement in software development[2]. The metric quality is often used to measuring the quality of software artifacts.

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 quality measurement of the class. The software consists of a lot of classes as the template or blueprint of the objects. The object can work together with other objects to accomplished the specific functionality of the software. The inner class measurement means measure relationships between elements inner the class (cohesion). The cohesion is important to inform us about the compactness of the class. The level of compactness of the class is related to external quality attributes named maintainability and understandability. The more level of compactness of class then the class is easier to change and understand.

Nowadays, several researchers are extensively discussed and evaluate software design problems named smell, in the other term is "code smell"[4], or "design flaw"[5]. The existence of smell may affect the quality attributes of the software.

The metric and the existence of smells indicate the quality of software artifacts. The metric can be used as a tool to measure quality. And, the existence of smell indicates lack of quality. The relationship between software metrics and smells is an interesting thing to learn. Bigonha et. al., explain the usefulness of the software metric threshold for detection of bad smells and fault prediction[6]. All matrices used by Bigonha are the class level's metric. The measurement is done on the source code level.

The position of smell becomes a guide for the developer to improve the software artifacts. The smell can be overcome by refactoring activity. Fowler defines the term refactoring as the process of changing internal structure that is not changing the external behavior of software[4]. The code is modified over time, and the structure of the code will gradually disappear. The bad smell is emerging and the quality of code is decay as stated in the Lehman's Laws that said as the software is evolving then 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. There is the researcher that concern about the importance of shift 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]. The challenge to do the smell detection on the design artifacts is the limitation of information. At the design level, there is only the model of the software that wants to build. One of the models as the product of the design is class diagram. Class diagram informs the structure of the software based on classes 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 is aimed to collect the information based on the syntactic and semantic information that exists inner the class diagram. It uses the information to detect the existence of the smell in the class diagram. And, it uses the inner information of class to detect the smell. Two types of smell that are considered related to cohesion are Blob and Feature Envy. There is information that will be extracted from the class diagram and then combine with the smell dataset that is taken from the Landfill dataset[11]. That information will be classified using three classifiers, j48, Multi-Layer Perceptron (MLP), and Naïve Bayes, and the result will be compared. The focus of this research is how the usefulness of the information can be used to detect the smell.

The rest of the paper is organized as follows. In section 2, we present every data that will be used in this research and how to get those data. Section 3 describes how to label every data. Section 4 describes the whole process of

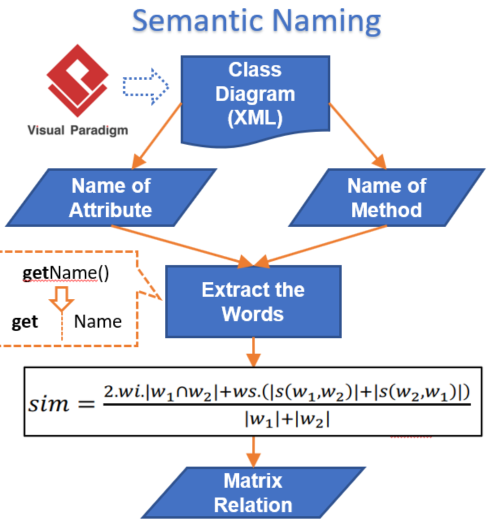

Fig. 1. Process of Semantic Similarity Analysis of the Label Name

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.

In collecting the data it is important to consider not only the syntactic information but also the semantic information. Syntactic information is the information that we can directly read or extract from the class diagram. And, the semantic information is the information that needs a little processing (interpret the meaning) to extract. The object of study in this research is the class diagram that resulted by using the Visual Paradigm tools. Then, it converts to the XML file then called XML based class diagram (Fig. 1).

2.1. *Number of Attributes*

The first data is the attribute. The collection of the attributes in the class diagram is easy. The notation for the attributes name and the type of attribute is clearly described in the class diagram.

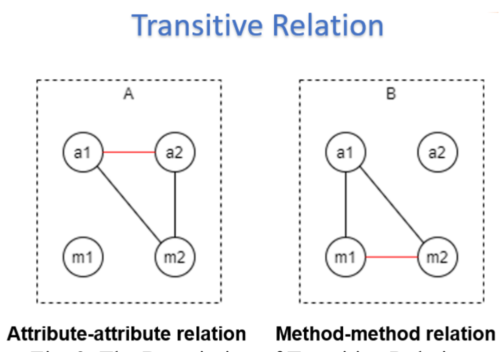

Fig. 2. The Description of Transitive Relation

The information about attributes is extracted from XML based class diagram.

2.2. *Number of Methods*

The second is the number of methods. The name, return value type and the parameters of the method are also clearly described in the class diagram. Collecting all information about the method is including in the syntactic information.

2.3. *Number of Relation between Method-Attribute*

The number of relations between Method and Attribute (MAR) can be count by considering not only syntactic but also semantic information. The relation between method and attribute is determined based on the similarity of type (syntactic) 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. All labels name are extracted from the XML's based class diagram. Then the labels are split into words. After that the similarity of meaning between the set of words is calculated[12]. If the label name between attribute and method is semantically similar then it will be considered as a relation. Matrix relation is generated 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 there are two methods that are related to the same attribute. Then, there will be a

Table 1. Representation of Dataset

The Seven Information Features

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

relation between attributes if there are two attributes that are related to the same method. This type of relation is included in the category of semantic information. Fig. 2 shows the analogy of both relations.

2.5. *Relation Capacity*

The maximum relation is the maximum number of relations that possible to exist in the class. The class is assumed as the area of the relation, that's has a capacity of relation. The maximum relation is calculated by using (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 is calculated by considering MAR, MMR, and AAR. The relation is direct relation and transitive relation. All of the relations between method and attribute is divided by the maximum relation (1) that possibly exists in one class. The cohesion calculation is expressed as (2).

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

3. Dataset

The Landfill dataset is consists of 243 instances of five types of code smells identified from 20 open-source software projects[11]. In the Landfill dataset, all classes that contain bad smell are labeled based on the type of smell.

In this research the dataset used to experiment is combine 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: jEdit, jHotDraw, FreeMind, HSQLDB, aTunes, and ArgoUML. The dataset consists of 300 data based on the smell class listed by the Landfill

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%

dataset and the smell free classes, then used as training data. And, the other set of 49 data is generated for testing data.

4. Classification

In this experiment, the process of detecting the smell will use the classification method. This research uses three classifiers: j48, Multi-Layer Perceptron, and Naïve Bayes. The tool used in this experiment is Weka as a Machine learning software to solve data mining problems[13]. All classifiers are run by using a basic configuration. The use of 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 explains the result of the experiment by using the dataset and three classifiers. The recapitulation of the result is described in Table 2.

j48 has a TPR of about 51%. 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%. MLP has a TPR value of about 51% of the data. Based on the label data, for Blob data is 94.7%, for Feature Envy data is 0% and for No data is 38.9%. Then, Naïve Bayes has a TPR value of about 44.9% where Blob data is 52.6%, Feature Envy data is 16.7% and No data is 55.6%.

Based on the result, 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 Naïve Bayes can detect the Feature Envy smell even though with the low rate. All classifiers can detect Blob smell with the 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. The existing information is not enough used to find both types of smells, especially for the Feature Envy smell. 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 dataset of class to identify the Blob smell in the class diagram is very effective. The effectiveness is proven by the TPR above 50%. On the other hand, it is not very good to use to identify the Feature Envy smell. The dataset does not express the big differential for Blob and Feature Envy. It makes the classifier hard to identify each type of smell. Only j48 and Naïve 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 important to find to sharpen the differences in the dataset.

References

1. Sommerville, I. *Software Engineering*. Software Engineering (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*. 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.