Research Article

Code Analysis Tool to Detect Extract Class Refactoring Activity in Vb.Net Classes

Sharif, K.Y[1], Muftah, M.G.H[2], Osman, M.H[3], Zakaria, M.L.[4]

[1,2,3,4]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Selangor, Malaysia
khaironi@upm.edu.my[1]

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: Code changes due to software change requests and bug fixing are inevitable in software lifecycle. The code modification may slowly deviate the code structure from its original structure that leads to unreadable code. Even though the code structure does not affect the software behaviour, it affects the code understandability and maintainability of software. Code refactoring is typically conducted to enhance the code structure; however, this task needs a lot of developers' effort. Thus, this paper aims at developing a tool that will help programmers identify possible code refactoring.Weconsider two aspects of refactoring:(i) refactoring activities, and (ii) refactoring prediction model. In terms of refactoring activity, we focus on Extract Class. The object-oriented metrics are used to predict the possibility of code refactoring. The combination of two refactoring aspects recommends the possible refactoring effort and identify classes that are involved. As a result, we managed to get 79% percent of accuracy based on the 11 correct results out of 14 that the tool correctly detected. On top of supporting programmers in improving codes, this work also may give more insight into how refactoring improvessystems.

Keywords: Data mining, Extract subclass, Object-oriented metrics, Refactoring, Refactoring detection

1. Introduction

Change requests and bugs fixing is one of the indicators that the software is still relevant to the business and customer. Software needs to be adaptable to new business or user requirements, and also the latest version of software needs to be delivered promptly. Slow response to a new requirement may lead to loss of business opportunity and profit loss. During the maintenance phase, the readability of source code is essential before code modification. Schnappinger etal. (2018) have shown that readability of software decay from time to time for various reasons. This legacy software is still required since it can perform the necessary functionality, and the product owner could not take the risk of migrating to new software. For this reason, software refactoring is an essential task to maintain software understandability as it preserves the readability aspect of the code, such as preserving code structure. Refactoring was defined by Fowler et al.(1999) as "*the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure*." This strategy envelops the accompanying four fundamental steps represented as(i) find where refactoring is needed,(ii) analyzing the cost and benefit of performing the refactoring,(iii) apply refactoring technique, and(iv) assuring the behavior of the product is not affected.One of the main ideas of refactoring is to avoid bad smells in the code design.Brown et al. (1998)argue that when centralized class hold all the responsibilities of system and the rest of the classes just hold data, complexity of a class like that led to the difficulty of reading the code which will affect the effort needed in the maintenance phase.

Extract Class refactoring is a technique to overcome the centralized class issue by splitting it into different classes that perform the same responsibilities without affecting its external behavior. The goal of this technique is to reach the high cohesion and low coupling of class design. However, extract class activities are tedious and require a lot of effort from developer and designer, Bavota et al. (2014). The extract class refactoring technique consists of five stages: (i) evaluating the core responsibility of each method in the class, (ii) group similar method in terms of responsibility, (iii) identifying methods that will hold the attributes of the class, (iv) divide methods and attributes in the separated class, and (v) ensure that the external behavior not affected.

In this paper, we proposed a code analysis tool to detect classes to be extracted in refactoring activities. We apply data mining techniques to produce the extract class refactoringprediction model. Object-oriented metrics such as Number of Methods (NOM), Lines of Code (LOC), and Number of Local Attributes (NOA) are used as the features for the prediction model. We use data that was collected by Al Dallal (2012), which consists of 2281 classes that are extracted from five different legacy software systems. The data from legacy systems were chosen since from our observation, the refactoring issue in legacy systems is more common and obvious. The contributions of this paper are the following: (i) we propose an extract class refactoring prediction model and (ii) we construct rule based on the prediction model for our code analysis tool. The rest of this paper organized as follows. Section 2 presents our related work, while Section 3 demonstrates the methodology of our research. Section 4 describes our findings on the usage of the tool. Finally, we conclude the paper and present the future work in Section 5.

2. Related Work

Many studies propose the prediction of class that possibly involve in refactoring activity such as Elish and Alsyayeb (2011); Hussain et al. (2020) and Ratzinger et al. (2007). The proposed approaches are commonly based on object-oriented metrics, precondition oriented, clustering oriented, code slicing, and dynamic analysis techniques.

Al Dallal (2012) introduced a prediction model to predict classes that likely to be involved in extract class activity. He investigated the capability of quality metrics in predicting the classes that probably be involved in refactoring exercise. Six Java-based products were used as a dataset and also the mutated classes. The study involved quality factors such as size, cohesion, and coupling.In terms of size, LOC (lines of code), NOM (number of local methods) and NOA (number of local attributes) were selected. For cohesion metrics,17 metrics were selected, such as Lack of Cohesion (LCOM) and Tight Class Cohesion (TCC). These selected metrics have been analyzed empirically to prove that they are useful to determine the classes in need of extract class refactoring.

In contrast, based on empirical investigation,Bavota et al. (2014) and Al-Kinani (2020) use several quality metrics such as Weighted Method per Class (WMC), Depth in Inheritance Tree and Number of Children (NOC) to show that compared with manual detecting of class in need of extract class operation those quality metrics do not have high accuracy. Deferent quality metrics could give a different result. Cohesion is one of the internal quality metrics attributes which might be affected by refactoring. Halim and Mursanto(2013) and Alshayeb(2009) present that there is no effect by applying refactoring operations on class cohesion.

Classification is the steps of finding a model that recognizes information classes, with the end goal of having the ability to predict the value of the class attribute that unknown. There are different types of classification algorithms like Decision Tree, Bayes and Rule-Based. More information on classification algorithms can be found in Han and Pei (2011).

3. Methodology

The overall development lifecycle is illustrated in Figure 1. The development starts with collecting software metrics to be used as the main dataset. The metrics were selected based on the recommendation from Al Dallal (2012). The analysis tool is built to compute the metrics that been selected previously, and then we used the data mining technique to constructa prediction model. Building the prediction model starts with data preprocessing.After that, we choose the best metrics to be used in the model and select a suitable algorithm that gives the best prediction accuracy. Then, we discover the rules based on the prediction model. The detail on the prediction model development is explained in the following:

A. Collect Data

The primary data of this project was derived from Al Dallal (2012). The dataset consists of 26 attributes that represent object-oriented software metrics for 2281 classes. These classes are derived from five different software systems. The last attribute called class is the result from the proposed model from Al Dallal (2012) that indicates whether the class in need of refactoring or not.

B. Choose the Appropriate Metrics

Al Dallal (2012) has proposed a mathematical model to measure whether a particular class in need of refactoring or not. Based on this work, we choose several object-oriented metrics, which are LOC, NOM, NOA, LCOM1, LCOM2, Coh, Response for a Class (RFC), and Data Abstraction Coupling (DAC1 and DAC2).

C. Implement the Analysis Tool

The analysis tool responsible for collecting selected code metrics and use the prediction model to recommend to the user regarding classes to be refactored. The tool consists of three main components:

a. Object-oriented Metrics component –This component is responsible for computing the metrics form the given code.
b. Prediction Model component - Build the prediction model and use it to predict the result.
c. User Feedback component - In this component, we collect the user feedback and add it to the dataset that

been used to construct the prediction model.

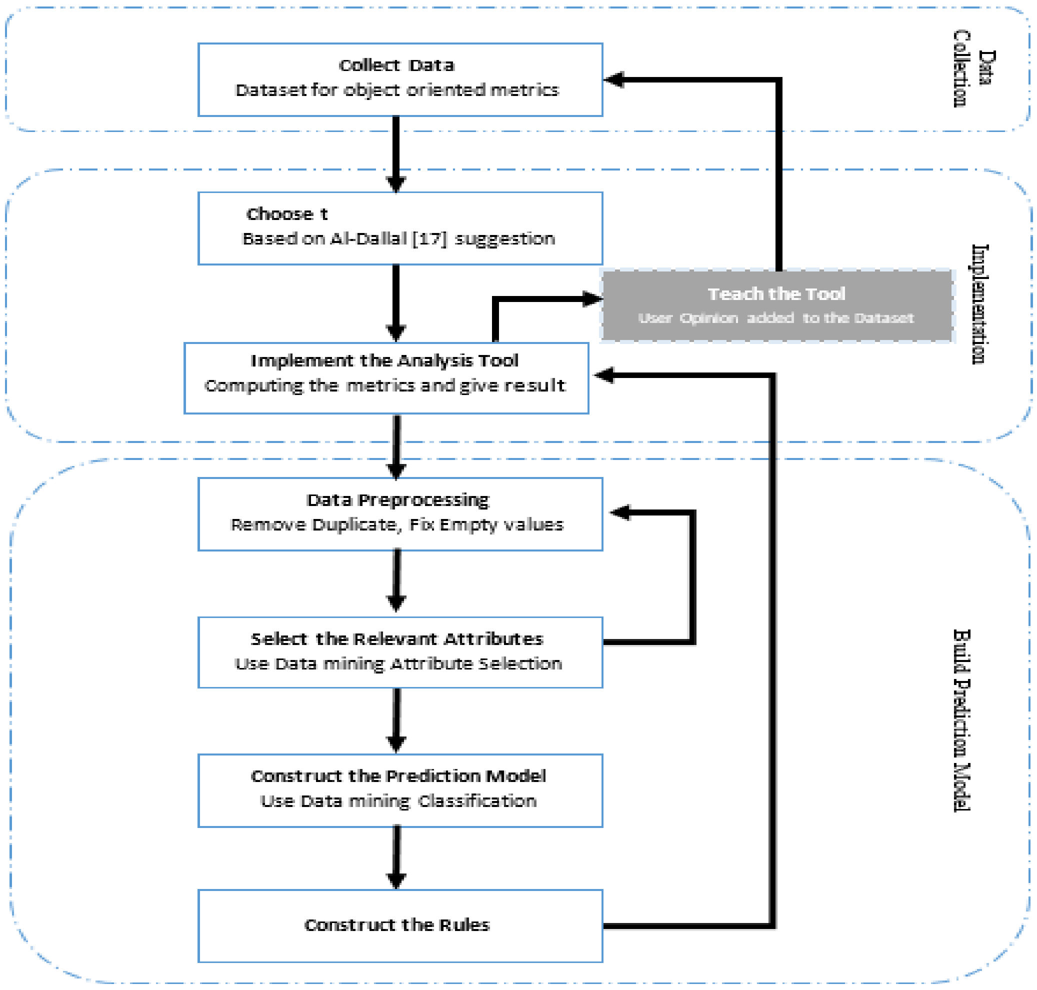
Collect Data
Dataset for object oriented metrics
Data Collection
Choose t
Based on Al-Dallal [17] suggestion
Teach the Tool
User Opinion added to the Dataset
Implementation
Implement the Analysis Tool
Computing the metrics and give result
Data Preprocessing
Remove Duplicate, Fix Empty values
Select the Relevant Attributes
Use Data mining Attribute Selection
Build Prediction Model
Construct the Prediction Model
Use Data mining Classification
Construct the Rules

Figure 1. Development Lifecycle

D. Data Processing

Data preprocessing describes any type of processing performed on raw data to prepare it for another processing procedure. In this context, we found that our dataset does not need any data preprocessing.

E. Select the Relevant Attributes

From our observation, not all selected attribute influences the prediction model. Hence, we perform attribute selection to eliminate attributes that do not hold or contain little information for prediction. By eliminating the fewer influence attributes, we may (i) improve the accuracy of the classification model, (ii) reduce the time for train the classification algorithm, and (iii) construct a simple prediction model. CfsSubsetEval feature selection algorithm is chosen since we have a dataset that its attributes are highly related to with the class attribute and poorly related to each other. The result shows that the attributes NOM, LCOM2, LCOM4, and Coh are appeared to be influential after performing this algorithm using the 10 cross-validations. Thus, these four attributes will be used to construct the prediction model.

F. Construct the Prediction Model

There are two steps involved in constructing the prediction model:

i. Selection of Classification Algorithm

There are some challenges to choose the algorithm to be used since a different algorithm constructs a different performance of the prediction model. Model selection is about to selecta suitable algorithm for a specific dataset. We use Waikato Environment for Knowledge Analysis (WEKA) to find a suitable algorithm for our dataset, Hall et al. (2009). WEKA environment experiment allows the user to evaluate multiple types of algorithms for our dataset. From this tool, we found that the J48 performs the highest accuracy (96.38%) in predicting the classes that need to be refactored.

ii. Model Construction

We use the following steps in constructing the model:

a. Dividing the dataset into two parts one for training represents 80% of the dataset, and the second one is the testdataset, which is 20%.

b. Perform a decision tree algorithm (J48) with the training dataset to train the algorithm to build the model, which gives us 97.7% accuracy.

c. Perform the decision tree algorithm (J48) using the test dataset to test the accuracy of the model with the dataset that not seen before, the accuracy is 97.6%.

d. The decision tree as shown in Figure 2, which will be converted to rules and then used in the proposed analysis tool to predict the result based on these four metrics.

G. Construct the rules

From the J48 decision tree, we extract the rules from the model (as illustrated in Figure 2). The extracted rules can be found in Table 1.

H. Teach the Tool

Teach the tool feature allows the user to give their comments about the result that has been given by the tool. We will further use the user comments to enhance the dataset in the future. This may improve the accuracy and reliability of the model.

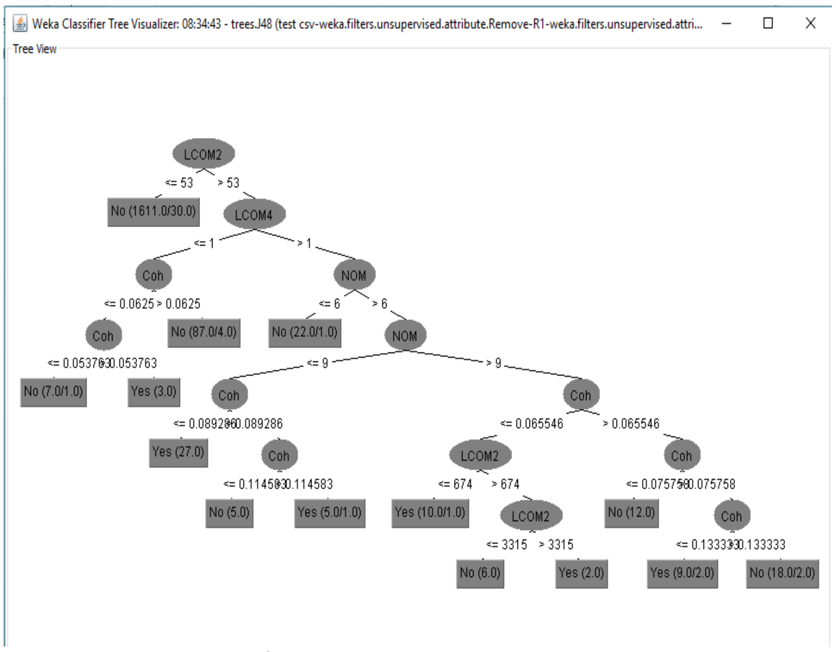

Figure 2. Example of Prediction model using decision tree algorithm

Table 1. Extracted Rules

No.	RULES	REFACTOR?
1.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh> 0.133	No
2.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh<= 0.133 and Coh<= 0.076	Yes
3.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh<= 0.076 and Coh<= 0.066	No
4.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh<= 0.066 and LCOM2 > 3315	Yes
5.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh<= 0.066 and LCOM2 <= 3315 and LCOM2 > 674	No
6.	LCOM2 > 53 and LCOM4 > 1 and NOM > 9 and Coh<= 0.066 and LCOM2 <= 674	Yes
7.	LCOM2 > 53 and LCOM4 > 1 and NOM <= 9 and Coh<= 0.115	Yes
8.	LCOM2 > 53 and LCOM4 > 1 and NOM <= 9 and Coh<= 0.115 and Coh> 0.089	No
9.	LCOM2 > 53 and LCOM4 > 1 and NOM <= 9 and Coh<= 0.115 and Coh<= 0.089	Yes
10.	LCOM2 > 53 and LCOM4 > 1 and NOM <= 6	No

11.	LCOM2 > 53 and LCOM4 <= 1 and Coh<= 0.063	No
12.	LCOM2 > 53 and LCOM4 <= 1 and Coh>0.54 and Coh<= 0.063	Yes
13.	LCOM2 > 53 and LCOM4 <= 1 Coh<= 0.054	No
14.	LCOM2 <= 53	No

4. Findings

To validate our proposed approach, we conduct an experiment by comparing classes that are selected to be refactored manually with the classes that are recommended based on our prediction model. Four postgraduate students with good working experience on VB.net performed the manual review on 14 VB.Net projects. The reviewers were asked to review these projects to find classes that need to be extracted, and then we compared the results that been obtained from the reviewers and the analysis tool.
From this experiment, we found 79% percent of accuracy based on the 11 correct results out of 14 that the tool correctly detected.

We also found that the attributes NOM, LCOM2, LCOM4, and Coh are the most relevant attribute that affected the decision to whether or not one class need to be extracted. However, in this initial experiment, we did not consider the time taken for the tool to complete the analysis and provide the result.We believe that having such a tool definitely will save time and effort for programmers to detect classes that need to extracted to improve code readability and maintainability.

5. Conclusion and Future Work

This paper has demonstrated an analysis tool that aims at supporting the programmer to detect class need to be extracted for refactoring,which could enhance the programmer's productivity. The tool was built to calculate four object-oriented metrics to be used as input of the prediction model. By recording the user opinion on the result, the ability of the prediction model can be improved. We evaluate the toll by conducting an experiment to compare the manual review of classes to be refactored and the extracted class based on our tool. This experiment involved four students and 14 VB.Net projects. We managed to get 79% percent of accuracy based on the 11 correct results out of 14 that the tool correctly detected.

For future work, we expect that the model may be improved by analysing the user comments on the outcome. Also, we are looking for other object-oriented metrics to be included as our attribute for prediction models. Conducting an extensive experiment to find the effectiveness of this tool compares to the manual is also our plan.

References

1. Alshayeb, M. (2009). Empirical investigation of refactoring effect on software quality. Information and software technology, 51(9), 1319-1326. Elsevier. Netherlands.
2. Al-Kinani, M.N.H., Adetunmbi, S.B., Hussain, A. (2020). Usability testing of mobile flipboard application on both non-users and novice users. International Journal of Interactive Mobile Technologies, 14 (5), pp. 47-56.
3. Bavota, G., De Lucia, A., Marcus, A., &Oliveto, R. (2014). Recommending Refactoring Operations in Large Software Systems. In Recommendation Systems in Software Engineering (pp. 387-419). Springer Berlin Heidelberg.
4. Brown, W. H., Malveau, R. C., & Mowbray, T. J. (1998). AntiPatterns: refactoring software, architectures, and projects in crisis. Wiley, USA
5. Elish, K. O., &Alshayeb, M. (2011). A classification of refactoring methods based on software quality attributes. Arabian Journal for Science and Engineering, 36(7), 1253-1267.
6. Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring: Improving the design of existing programs. Addison-Wesley Professional. USA
7. Halim, A., &Mursanto, P. (2013, October). Refactoring rules effect of class cohesion on high-level design. In Information Technology and Electrical Engineering (ICITEE), 2013 International Conference on (pp. 197-202). IEEE. USA
8. Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques. Elsevier. Netherlands.
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18. USA.
10. Hussain, A., Subramanian, G. (2020). A bibliometric research analysis for online booking. Journal of Advanced Research in Dynamical and Control Systems, 12 (6), pp. 1671-1679.

11. Ratzinger, J., Sigmund, T., Vorburger, P., & Gall, H. (2007). Mining software evolution to predict refactoring. In Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on (pp. 354-363). IEEE. USA.
12. Schnappinger, M., Osman, M. H., Pretschner, A., Pizka, M., &Fietzke, A. (2018). Software quality assessment in practice: a hypothesis-driven framework. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-6). ACM. USA.