I.J. Information Engineering and Electronic Business, 2020, 4, 21-29
Published Online August 2020 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2020.04.03

Reduction of Multiple Move Method Suggestions Using Total Call-Frequencies of Distinct Entities

Atish Kumar Dipongkor
Department of Computer Science and Engineering
Jashore University of Science and Technology, Bangladesh
Email: atish.cse@just.edu.bd

Rayhanul Islam
Institute of Leather Engineering and Technology, University of Dhaka
Email: rayhanul.islam@du.ac.bd

Nadia Nahar, Iftekhar Ahmed, Kishan Kumar Ganguly, S.M. Arif Raian, Abdus Satter
Institute of Information Technology, University of Dhaka
Email: nadia@iit.du.ac.bd, iftekhar.ahmed@rwth-aachen.de, kkganguly@iit.du.ac.bd, arif0224@gmail.com, abdus.satter@iit.du.ac.bd

Received: 21 March 2020; Accepted: 24 June 2020; Published: 08 August 2020

Abstract: Inappropriate placement of methods causes Feature Envy (FE) code smell and makes classes coupled with each other. To achieve cohesion among classes, FE code smell can be removed using automated Move Method Refactoring (MMR) suggestions. However, challenges arise when existing techniques provide multiple MMR suggestions for a single FE instance. The developers need to manually find an appropriate target classes for applying MMR as an FE instance cannot be moved to multiple classes. In this paper, a technique is proposed named *MultiMMRSReducer*, to reduce multiple MMR suggestions by considering the Total Call-Frequencies of Distinct Entities (TCFDE). Experimental results show that TCFDE can reduce the multiple MMR suggestions of an FE instance and performs 77.92% better than an existing approach, namely, JDeodorant. Moreover, it can ensure minimum future changes in the dependent classes of an FE instance.

Index Terms: Feature Envy, Move Method Refactoring.

1. Introduction

The expected behavior and characteristics of object-oriented (OO) software design is low coupling and high cohesion among software components [1, 2, 26, 27]. Although these are important characteristics, Feature Envy (FE), considered as one of the code smells, usually makes software more coupled and lower cohesive which eventually ensues less maintainable software [8, 9, 26, 27]. Moreover, it increases software's fault-proneness and decreases productivity and rework of a software [3, 4, 5, 6, 7]. FE occurs in source code when an entity (method or attribute) uses/envies too many entities of other classes to obtain data and/or functionality [10]. Considering the impact of high coupling and low cohesion in software maintenance, FE instances should be identified and eliminated from source code through Move Method Refactoring (MMR) techniques [8, 9, 10, 11, 12, 29]. MMR is applied to move an FE instance from its source class to the envied class for ensuring low coupling and high cohesion among components. However, the main problem is when an FE instance envies the equal number of entities from multiple classes. In such situations, the developer cannot choose appropriate envied or target class to apply MMR automatically. For example, *sendSingleMessage()* (in **Fig. 1**) is an FE instance as it envies more entities from the class *TP* and *ByteArrayDataOutputStream* than its source class *NoBundler*. For this reason, *sendSingleMessage()* can be moved either to *TP* or *ByteArrayDataOutputStream*. However, there is a hidden maintenance cost associated with each target class that involves the frequency of entity calls (3 and 6 respectively) from *TP* and *ByteArrayDataOutputStream*. The FE instance *sendSingleMessage()* calls methods from *ByteArrayDataOutputStream* more frequently and thus it is more coupled with *ByteArrayDataOutputStream* than TP. Now, if *sendSingleMessage()* is moved to TP and changes occur in the three distinct methods of *ByteArrayDataOutputStream*, it needs to be modified six times in *TP* due to the higher call frequencies (6>3). On the contrary, if *sendSingleMessage()* is moved to *ByteArrayDataOutputStream* and changes

occur in the three distinct methods of *TP*, it needs to be modified three times in *ByteArrayDataOutputStream*. To this end, a technique is required to find an appropriate target class for the FE instances like *sendSingleMessage()*.

Considering maintenance issues, it is needed to apply MMR based on the frequency of entity calls for above scenario. To the best of the author's knowledge, there is no automated technique to alleviate this problem. Although developers can manually check the frequency of entity calls while refactoring, it is a time-consuming in terms of software maintenance. Thus, an automated technique by considering the frequency of entity calls will be helpful to resolve this problem. However, it is challenging to automatically identify the frequency of entity calls for FE instances, because they call/use entities from both built-in and used-defined classes. As it is not possible to move FE instances in the built-in classes, only user defined classes should be considered for checking the frequency of entity calls.

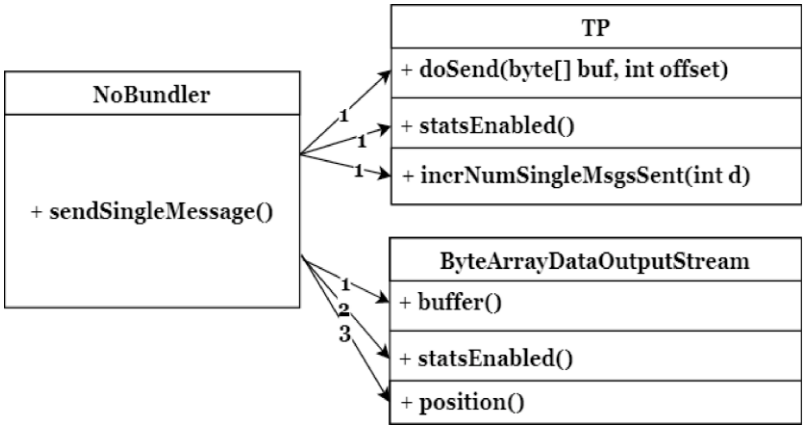

Fig. 1. Multiple MMR target classes for an FE instance

Several automated techniques provide MMR suggestions to move FE instances in the suitable target class. However, these techniques cannot provide appropriate MMR suggestion for the FE instances those call/envy equal number of entities from different classes. Basically, these techniques provide multiple MMR suggestions in these kinds of scenarios which requires manual interaction from developer to find an appropriate MMR suggestion. Simon *et al.* [13] used distance-based cohesion metric for identifying MMR opportunities in small applications. Tsantalis *et al.* [14] proposed an improved technique named JDeodorant with compared to distance-based cohesion metric. The technique identifies FE instances based on their entity usages from other classes. Another approach is proposed by Jehad Al Dallal [15] to predict MMR opportunities in object-oriented software systems. This approach considered both coupling and cohesion aspects of methods for predicting MMR scopes. There are some other techniques, for example, JMove, that provide MMR suggestions based on similarity measurements. JMove [16] provides MMR suggestions by measuring the similarity coefficient of dependency sets. MMRUC3 [17] provides MMR suggestions using Jaccard and Contextual similarities of the dependency set. Methodbook [18] identifies MMR opportunities using Relational Topic Models. TACO [19] provides MMR suggestions based on textual similarities of methods. HIST [20] identifies FE instances and provides MMR suggestions based on historical co-changes. All the techniques mentioned here provide promising MMR suggestions except the case - when an FE instance calls/envies equal number of entities from different classes.

In this paper, a technique is proposed, named *MultiMMRSReducer* to reduce multiple MMR suggestions of an FE instance. The main aim of this technique is to find an appropriate target class as it is not possible to move an FE instance in multiple classes. At first, a method is considered as an FE instance and MMR suggestion(s) is generated using an existing technique [l4] if it calls/envies more distinct entities from other classes that its own. Then, all the FE instances having multiple MMR suggestions are separated. After that, for each FE instance, the frequency of calls to distinct entities in each envied class is calculated. Finally, the most frequently called envied class is selected as an appropriate MMR suggestion.

To evaluate the proposed technique, an empirical study is conducted on nine open source Java projects. The experimental results show that the proposed technique can avoid multiple MMR suggestions for an FE instance with compared to an existing approach named JDeodorant [14]. Moreover, it is found that MMR suggestions of the proposed technique can ensure minimum number of changes in the FE instances.

2. Literature Review

Due to coupling and cohesion issues, Move Method Refactoring (MMR) has been drawing researchers' attention for many years. In this section, existing works on MMR are discussed with their strengths and weaknesses. Moreover, these works are grouped into two categories based on their methodologies such as (a) Identifying MMR suggestions using Coupling and Cohesion and (b) Identifying MMR suggestions based on Similarity [28].

2.1 Identifying MMR suggestions using Coupling and Cohesion

The existing works discussed here identify FE instances based on the Degree of Coupling. Besides, they provide MMR suggestions to increase cohesion and reduce coupling among software components. Although the MMR suggestions of this techniques are promising, some major issues are observed which indicate opportunities for further work. In order to identify Move Method and attribute scopes, Simon *et al.* [13] have proposed a distance-based cohesion metric. According to this technique, an entity (method or attribute) should be moved if it uses or used by other entities of the different classes than its source class. In real world, an entity can use or be used by equal number of entities from different classes. Thus, it is not clear from this approach which class is appropriate for moving that entity. Moreover, this approach does not examine behavior-preserving [21] preconditions before identifying Move Method and attribute situations.

With compared to Simon *et al*. [23], Tsantalis *et al*. [14] have proposed an improved technique named JDeodorant for identifying MMR opportunities in source code. Their technique treats a method *m* as an FE instance if it calls/uses more entities from class C_i than its current class *C*. Then, it suggests moving m in C_i if m meets some behavior-preserving preconditions (Compilation, Behavior-Preservation and Quality). However, their technique suggests multiple classes to move m if m calls/uses equal number of distinct entities from different classes. Although their technique evaluated maintenance effects (in terms of coupling and cohesion) of MMR suggestions using entity placement metric, it did not provide any direction for multiple suggestions issue for a single FE instance.

Jehad Al Dallal [15] used a statistical technique to predict MMR opportunities in Object-Oriented systems. Both coupling and cohesion aspects of methods are considered for identifying MMR scopes. However, there is no direction about predicting MMR opportunities when a method becomes equally coupled with multiple classes. Besides, the author claimed high precision of his technique but there is no comparative study with the existing techniques.

2.2 Identifying MMR suggestions based on Similarity

All the techniques discussed in here provide MMR suggestions using some similarity measures, i.e., structural, semantical and so forth. Under these techniques, a method should be moved if it is more like another class than its source class. To the best of our knowledge, these techniques include problems of providing multiple MMR suggestions for a single FE instance.

Using similarity coefficient [23, 24] of dependency sets, JMove [16] provides MMR suggestions to group similar methods together. Firstly, for a given method *m*, it calculates similarity coefficient with all methods of its current class. Then, it calculates similarity coefficient with rest of the methods in other classes. If the similarity coefficient is higher in a class other than its current class and *m* satisfies all behavior-preserving preconditions proposed by JDeodorant [14], this approach provides MMR suggestion for *m*. The major issues of this technique are: (a) it provides multiple MMR suggestions for a method when the similarity coefficients become equal with multiple classes and (b) its MMR suggestions contain a high percentage of false-positive compared to JDeodorant.

MMRUC3 [17] is a JMove-like technique which also provide MMR suggestions for eliminating FE instances. The main differences between MMRUC3 and JMove are - MMRUC3 uses a different similarity measurement (Jaccard Similarity), and consideration of contextual similarity along with the dependency sets similarity. With compared to JMove, this approach introduced two new things, but it did not provide any justification for that. Moreover, there is no direction for the methods having equal similarity coefficient with multiple classes.

Methodbook [18] identifies MMR opportunities using Relational Topic Models (RTM) [25] for removing FE instances. According to this approach, a method *m* should be moved to another class C_i if C_i contains more friends of *m* than its own class *C*. To find friends of the method *m*, Methodbook considers three types of similarity with the methods of other classes such as (1) similarity in using same attributes, (2) similarity in calling same methods and (3) semantic similarity between methods. From this approach, it is not clear where a method should be moved when it contains equal number of friends in multiple classes. Moreover, the finding of Methodbook showed that it could not perform better than JDeodorant.

TACO [19] gives MMR recommendations to eradicate FE instances based on the textual similarity of methods. If a method *m* is textually like another class C_i than its current class *C*, then *m* should be moved to C_i. In this approach, Cosine Similarity is used to measure the Textual similarity among methods. The main issues of this approach are (a) it provides multiple MMR suggestions when similarity becomes equal with several classes, (a) it does not examine any behavior-preserving preconditions before providing MMR suggestions and (c) it cannot provide accurate MMR suggestions without enough textual information.

HIST [20] provides MMR suggestions based on historical co-changes for eliminating FE instances from source code. According to this approach, if a method *m* changes more frequently when changes occur in its dependent class C_i then it should be moved to C_i. The main limitations of this approach are (a) it cannot provide accurate MMR suggestions without enough historical information (b) it does not consider any structural information (i.e., whether *m* calls any methods and/or attributes from C_i or not) and (c) there is no guidance where a method should be moved if historical co-changes occur in the method and several classes at the same time.

3. Proposed Technique

The proposed technique aims at finding the appropriate target class automatically for an FE instance and ensuring minimum future changes in other dependent classes of the instance. Consider, a method *m* calls/uses equal number of distinct entities from two different classes, for example, $E_1 = \{e_1, e_2\}$ from class C_1 and $E_2 = \{e_3, e_4\}$ from C_2. According to the existing approaches, any of these two classes (C_1 or C_2) can be the target class for applying MMR because m is equally coupled [$n(E_1) = n(E_2)$] with both classes. However, there is a maintenance issue when total call frequencies of these distinct entities vary between C_1 and C_2. For instance, F_1 and F_2 are the total number of calls to all members of E_1 and E_2 where F_1 is less than F_2. As *m* uses entities from C_2 more frequently ($F_1 < F_2$), we can say it is more coupled with C_2 than C_1. That being said, if someone ignores this total call frequencies and move *m* to C_1, future change rate will be higher in *m* (if all members of E_2 change). On the other hand, if *m* is moved to C_2 and changes occur in all members of E_1, then change rate will be less in *m*. To this end, a formula is defined for calculating Total Call-Frequencies of Distinct Entities (TCFDE) between a method *m* and target class *C*.

$$\mathrm{TCFDC(m, c)} = \sum_{i=1}^{n} CFDE_i \quad (1)$$

Where *m* = FE instance, *C* = given target class, n = Total distinct entities *m* uses from *C*, $CFDE_i$ = total number of times i^{th} distinct entity of *C* is used by *m*.

Using Equation (1), an algorithm is proposed for finding appropriate target class for an FE instance which is equally coupled multiple classes. Initially, the proposed Algorithm requires an FE instance and its target classes as input to find an appropriate target class for it. In order to identify FE instances and their respective target classes from the source code, an existing Algorithm [14] is used which consists of following steps:

1. Identification of candidate target classes *T* from where a method *m* uses/calls more entities than its current class.
2. Examination of preconditions for each class of *T* such as compilation, behavior-preservation and quality. This step ensures the applications behaviors if m is moved to any class of *T*.
3. Examination of whether method m modifies a data structure in each class of *T*. This step ensures strong conceptual binding with a specific target class.
4. Filtering *T* based on step #2 and #3. If a class of *T* satisfies all preconditions, it will be considered as MMR target class and *m* is identified as an FE instance.

After identifying FE instances and their respective MMR target classes using existing Algorithm [14], only the FE instances having multiple MMR target classes is sent to the proposed *MultiMMRSReducer* algorithm one by one. The aim of the proposed algorithm is to find appropriate target class A_{tc} for each FE instance. Initially, A_{tc} is empty (in line #2). Then, a variable MAX_{TCFDE} is defined to store maximum *TCFDE* between an FE instance *m* and its target classes in line #3. Finally, it is iterated over the list of target classes of *m* (line #4 to #9). In each iteration, it is verified whether the *TCFDE* between *m* and current iterated class *t* is greater than MAX_{TCFDE}. If it is found that *TCFDE (m, t)* is greater than MAX_{TCFDE}, A_{tc} is updated with *t* and MAX_{TCFDE} is updated with *TCFDE (m, t)*. After the final iteration, the value of A_{tc} is selected as the appropriate target class for *m*.

4. Implementation and Result Analysis

For comparative result analysis, the proposed Algorithm was implemented as a software tool (Eclipse-plugin), *MultiMMRSReducer*, using Java. Then, an empirical study was conducted on seven open-source projects from Github to evaluate the effectiveness of *MultiMMRSReducer*. The detailed description of the dataset[1], i.e., Lines of Code (LOC), Number of Classes (NOC), Number of Methods (NOM), etc. is shown in Table 1. Then, to show the effectiveness, the developed software tool- *MultiMMRSReducer* was compared with an existing tool named JDeodorant.

Table 1. The dataset used in this study.

Project	LOC	NOC	NOM
base spring	2579	44	147
greenhouse	27376	440	1703
jgroup	103081	770	8186
SpringBlog	2241	46	185
SpringMVCDemo	457	6	53
spring-petclinic	2527	37	144
greenmail	22658	268	1386

[1] https://bit.ly/2IUqku0

In Table 1, 1st column: Name of the Projects, 2nd column: LOC - Lines of code in each project, 3rd column: NOC- Number of Classes in each project, 4th column: NOM- Number of Methods in each project.

4.1 Environmental Setup

This section outlines the required steps for conducting the experimental analysis of the proposed algorithm. At the initial step, multiple MMR suggestions from the dataset (Table 1) were identified to examine if *MultiMMRSReducer* can reduce these suggestions automatically or not. The identification phases of multiple MMR suggestions were performed by following a set steps which are described below-

1. At first, Move Method Refactoring (MMR) suggestions from the dataset were computed using the JDeodorant [14] Eclipse Plug-in.
2. Next, the FE instances having multiple MMR suggestions were manually separated. The Table 2 contains project-wise multiple MMR suggestions identified manually.

After identifying the list of multiple MMR suggestions manually, Number of Distinct Entities (NDE) used/called by an FE instance from each target class and Total Call-Frequencies of Distinct Entities (TCFDE) in each target class are calculated by analyzing the source code. This information is represented in 6th and 7th column of Table 2, respectively. Then, these FE instances are grouped in two categories based on the values of *TCFDE* in each target class such as (a) Equal TCFDE and (b) Not Equal TCFDE. This categorization is performed for evaluation purpose of the proposed technique.

Equal TCFDE: If the *TCFDE* between an FE instance and all target classes are equal, then the instance is added to this category. From Table 2, the FE instances with ID #5, #7, #9 and #14 were found in this category.

Not Equal TCFDE: If the *TCFDE* between an FE instance and all target classes are not equal, then the instance is added to this category. From Table 2, the FE instances with ID #1, #2, #3, #4, #6, #8, #10, #11, #12 and #13 were found in this category.

4.2 Evaluation and Result Discussion

To evaluate the proposed technique, this study answers the following research questions–

1. **RQ #1:** Is it possible to reduce multiple MMR suggestions using *TCFDE*?
2. **RQ #2:** How much multiple MMR suggestions of JDeodorant is reduced by the *MultiMMRSReducer* effectively?

The above research questions are answered by examining if *MultiMMRSReducer* can reduce all the multiple suggestions collected by the developers (in Table 2). To that end, the *MultiMMRSReducer* is applied on the dataset. 2nd column of Table **3** shows the number FE instances having multiple MMR suggestions in each project.

Table 2. Project wise multiple MMR suggestions for a single FE instance.

Id	Project	Method	Source Class	Target Class	NDE	TCFDE
1	jgroup	sendSingleMessage	NoBundler	ByteArrayDataOutputStream	3	6
				TP	3	3
2	jgroup	createMessage	Route	Protocol	2	4
				Message	2	2
3	jgroup	removeMessage	DeliveryManagerImpl	MessageInfo	4	4
				Message	4	6
4	jgroup	setHeader	ForkChannel	Message	3	5
				GossipData	3	4
5	SpringBlog	updateSettings	AdminController	SettingsForm	3	3
				AppSetting	3	3
6	base spring	credentialsMatch	CredentialValidation	User	2	3
				UserDAO	2	2
7	greenhouse	settingsPage	SettingsController	Account	5	6
				Profile	5	6

8	greenmail	copyHeaders	TopCommand	Pop3Connection	2	6
				ImapRequestLineReader	2	3
9	greenmail	copyLines	TopCommand	Pop3Connection	4	4
				InternetPrintWriter	4	4
10	greenmail	consumeWord	CommandParser	ImapRequestLineReader	3	4
				Partial	3	3
11	greenmail	nextNonSpaceChar	FetchCommandParser	ImapRequestLineReader	2	5
				SmtpConnection	2	2
12	SpringBlog	createSpringUser	UserService	User	2	4
				SettingsForm	2	3
13	Spring petclinic	addVisit	Pet	Owner	3	3
				Visit	3	4
14	SpringMVCDemo	addBlogPost	BlogController	BlogEntity	2	2
				UserEntity	2	2

Answer to RQ #1: As *MultiMMRSReducer* reduces multiple MMR suggestions based on the value of *TCFDE* between an FE instance and a target class, it is required to check its performance on both Equal (a) and Not Equal *TCFDE* (b) Categories. For Not Equal *TCFDE* Category (b), *MultiMMRSReducer* performed accurately. Basically, it finds appropriate target class for all FE instances. The reason is that the values of *TCFDE* were different in each target class, and the highest *TCFDE* valued target class was selected as appropriate one. For example, the FE instance *sendSingleMessage()* with ID #1 in **Table 2** has two target classes namely, *ByteArrayDataOutputStream* and *TP* as the value of NDE is equal in both target classes. The *MultiMMRSReducer* was able to find appropriate target class *ByteArrayDataOutputStream* as the *TCFDE (sendSingleMessage, ByteArrayDataOutputStream)* is greater than *TCFDE (sendSingleMessage, TP)*.

Algorithm: *MultiMMRSReducer* - Finding Appropriate Target Class for an FE instance having multiple MMR suggestions

Input: An FE instance *m* and List of target classes *List*<T_c> of it. Here, *m* is equally coupled with the all classes in list *List*<T_c>.
Output: Appropriate target class A_{tc} of m for applying MMR. Ensuring that *TCFDE* between *m* and A_{tc} will be maximum.

1. *function findAppropriateTargetClass(m, List*<T_c>*)*
2. A_{tc} = ø
3. MAX_{TCFDE} = 0
4. *for t in List*<T_c> *do:*
5. *if TCFDE (m, t) >* MAX_{TCFDE} *then*
6. A_{tc} = *t*
7. MAX_{TCFDE} = *TCFDE (m, t)*
8. *end if*
9. *end for*
10. *return* A_{tc}
11. *end function*

Similarly, it selected appropriate target classes *Protocol*, *MessageInfo*, *Message*, *User*, *Pop3Connection*, *ImapRequestLineReader*, *ImapRequestLineReader*, *User* and *Owner*, respectively, for the FE instances with ID #2, #3, #4, #6, #8, #10, #11, #12 and #13. Although *MultiMMRSReducer* shows its best performance for Not Equal TCFDE Category (b), it could not reduce multiple MMR suggestions for Equal TCFDE Category (a) because the values of TCFDE were equal in each target class. For example, the FE instance *addBlogPost* with ID #14 in Table 2 has two MMR suggestions such as (*addBlogPost* → *BlogEntity*) and (*addBlogPost* → *UserEntity*). Since the *TCFDE (addBlogPost, BlogEntity)* is equal to *TCFDE (addBlogPost, UserEntity)*, *MultiMMRSReducer* could not find appropriate target class for *addBlogPost*. Similarly, *MultiMMRSReducer* miss stepped for the FE instances with ID #5, #7, #9 of Equal TCFDE Category (a).

In Table 2, 1st column represents the ID of FE instances, 2nd column represents the project names, 3rd column represents FE instances, 4th column represents the source class of the FE instances, and 5th column represents the target classes of the FE instances.

Table 3. Project wise multiple MMR suggestions reduced by the proposed technique.

Project	Multi MMRS of JDeodorant	Proposed MultiMMRSReduced
spring-petclinic	1	1
SpringBlog	1	1
greenhouse	1	0
jgroups	4	4
SpringMVCDemo	1	0
greenmail	4	3
base spring	1	1

Answer to RQ#2: The *MultiMMRSReducer* was able to reduce almost all the FE instances having multiple MMR suggestions, i.e., 3rd column of Table 3 shows the number FE instances for those multiple suggestions is reduced by *MultiMMRSReducer*. The 2nd column of Table 3 shows the number of FE instances having multiple MMR suggestions which are given by the JDeodorant.

From this Table 3, it can be observed that *MultiMMRSReducer* reduced multiple MMR suggestions for the all FE instances from all projects in our dataset except SpringMVCDemo, greenmail and greenhouse. The reason of not reducing multiple MMR suggestions is that the values of *TCFDE* were equal in each target class. For our dataset, JDeodorant provided multiple MMR suggestions for thirteen FE instances whereas *MultiMMRSReducer* eliminated these multiple suggestions for ten FE instances. By comparing this statistic, it can be said that *MultiMMRSReducer* shows performance 77.92 percentage better than JDeodorant. Apart from this, the *MultiMMRSReducer* can ensure minimum future changes inside the FE instances. Basically, the FE instances will go through minimum future changes if refactoring is performed based on the selected appropriate target class by *MultiMMRSReducer*. For example, *sendSingleMessage* is moved to *TP*, and changes (i.e., addition or deletion of parameters) occur to all envied methods of *ByteArrayDataOutputStream*. Then, *sendSingleMessage* needs to be changed six times *(TCFDE (sendSingleMessage ByteArrayDataOutputStream) = 6)*. On the other hand, if *sendSingleMessage* is moved to *ByteArrayDataOutputStream* according to *MultiMMRSReducer*, and changes occur to all envied methods of *TP*. Then, the number of modifications in *sendSingleMessage* will be three times *(TCFDE (sendSingleMessage, TP) = 3)*. As *MultiMMRSReducer* ensures minimum change occurrences inside the FE instances, it will help developers to write more maintainable and less changeable code.

In Table 3, 1st column: Name of the project, 2nd column: Number of FE instances having multiple MMR suggestions, 3rd column: Number of FE instances for those multiple suggestions is reduced by the proposed *MultiMMRSReducer* technique.

5. Conclusion

According to the existing techniques, a Feature Envy (FE) instance can be moved to its any equally coupled target classes. However, there is a hidden maintenance cost associated with moving to each target class. To address this issue, a new technique is proposed which can find an appropriate target class for an FE instance using Total Call-Frequencies of Distinct Entities (TCFDE). To evaluate, an empirical study is conducted on nine open source Java projects. The experimental results show that the proposed technique can avoid multiple target classes 77.92% better than an existing technique named JDeodorant. As the future work, the feedbacks from the software developers will be added to the prposed techniques. Moreover, the performance and reliability of the proposed technique from the developers' perspective will be analyzed.

References

[1] E. Gamma, Design patterns: elements of reusable object-oriented software, Pearson Education India, 1995.
[2] R. C. Martin, Agile software development: principles, patterns, and practices, Prentice Hall, 2002.
[3] V. R. Basili, L. C. Briand and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Transactions on software engineering, vol. 22, p. 751–761, 1996.
[4] L. C. Briand, J. Wüst, S. V. Ikonomovski and H. Lounis, "Investigating quality factors in object-oriented designs: an industrial case study," in Proceedings of the 21st international conference on Software engineering, 1999.
[5] S. R. Chidamber, D. P. Darcy and C. F. Kemerer, "Managerial use of metrics for object-oriented software: An exploratory analysis," IEEE Transactions on software Engineering, vol. 24, p. 629–639, 1998.
[6] L. C. Briand, J. Wust and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, 1999.

[7] M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman and G. Saint-Denis, "Design Properties and Object-Oriented Software Changeability," in 4th European Conference on Software Maintenance and Reengineering, CSMR 2000, Zurich, Switzerland, February 29 - March 3, 2000., 2000.
[8] M. D'Ambros, A. Bacchelli and M. Lanza, "On the impact of design flaws on software defects," in Quality Software (QSIC), 2010 10th International Conference on, 2010.
[9] D. I. K. Sjøberg, A. Yamashita, B. C. D. Anda, A. Mockus and T. Dybå, "Quantifying the effect of code smells on maintenance effort," IEEE Transactions on Software Engineering, vol. 39, p. 1144–1156, 2013.
[10] M. Fowler and K. Beck, Refactoring: improving the design of existing code, Addison-Wesley Professional, 1999.
[11] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of OO Systems, Springer, 2006.
[12] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?," in Software Maintenance (ICSM), 2012 28th IEEE International Conference on, 2012.
[13] F. Simon, F. Steinbruckner and C. Lewerentz, "Metrics based refactoring," in Software Maintenance and Reengineering, 2001. Fifth European Conference on, 2001.
[14] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," IEEE Transactions on Software Engineering, vol. 35, p. 347–367, 2009.
[15] J. A. Dallal, "Predicting move method refactoring opportunities in object-oriented code," Information and Software Technology, vol. 92, p. 105–120, 2017.
[16] V. Sales, R. Terra, L. F. Miranda and M. T. Valente, "Recommending move method refactorings using dependency sets," in Reverse Engineering (WCRE), 2013 20th Working Conference on, 2013.
[17] R. M. Masudur, R. R. Rubby, K. S. Mostafa, S. Abdus and R. M. Rayhanur, "MMRUC3: A recommendation approach of move method refactoring using coupling, cohesion, and contextual similarity to enhance software design," Softw Pract Exper. 2018, pp. 1-28.
[18] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk and A. De Lucia, "Methodbook: Recommending move method refactorings via relational topic models," IEEE Transactions on Software Engineering, vol. 40, p. 671–694, 2014.
[19] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto and A. Zaidman, "A textual-based technique for smell detection," in Program Comprehension (ICPC), 2016 IEEE 24th International Conference on, 2016.
[20] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in Automated software engineering (ASE), 2013 IEEE/ACM 28th international conference on, 2013.
[21] W. F. Opdyke, "Refactoring object-oriented frameworks," 1992.
[22] T. Mens and T. Tourwé, "A survey of software refactoring," IEEE Transactions on software engineering, vol. 30, p. 126–139, 2004.
[23] R. R. Sokal and P. H. A. Sneath, "Principles of numerical taxonomy WH Freeman," San Francisco, CA, 1963.
[24] P. H. A. Sneath, R. R. Sokal and others, Numerical taxonomy. The principles and practice of numerical classification., 1973.
[25] J. Chang and D. Blei, "Relational topic models for document networks," in Artificial Intelligence and Statistics, 2009.
[26] R. Islam and K. Sakib A package-based clustering approach to enhance the accuracy and performance of software defect prediction. International Journal of Software Engineering, Technology and Applications. 2017; 2(1):1-21.
[27] R. Islam, and K. Sakib. "A Package Based Clustering for enhancing software defect prediction accuracy." In 2014 17th International Conference on Computer and Information Technology (ICCIT), pp. 81-86. IEEE, 2014.
[28] A. K. Dipongkor , I. Ahmed, and N. Nahar. "Move Method Recommendation using Call Frequency of Methods and Attributes." In 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR), pp. 76-81. IEEE, 2018.
[29] M. Selim, S. Siddik, A. U. Gias, M. Wadud, and S. M. Khaled. "A genetic algorithm for software design migration from structured to object oriented paradigm." arXiv preprint arXiv:1407.6116 (2014).

Authors' Profiles

Atish Kumar Dipongkor is a faculty member of Computer Science and engineering at Jashore University of Science and Technology (JUST), Bangladesh. He has earned his Master of Science in Software Engineering (MSSE) from the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. Before joining JUST as a lecturer, he has worked as a senior software engineer in a multinational IT organization (Brain Station 23 Ltd.). His core areas of interest are Code Smell, Refactoring, System Architecture Design, Web Technologies, and Bangla Text Processing.

Rayhanul Islam is currently working as a Lecturer at the Institute of Leather Engineering and Technology (ILET), University of Dhaka, Dhaka, Bangladesh. He has completed Master of Science in Software Engineering (MSSE) with the thesis in Software source code's dimension reduction for defect prediction from Information Technology (IIT), University of Dhaka. He also worked as Associate Software Engineer at a renowned software company named KAZ Software Ltd. His research interests include Software Engineering, Machine Learning, and Data Mining.

Nadia Nahar is Lecturer at the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. She pursued her Master of Science in Software Engineering (MSSE) and Bachelor of Science in Software Engineering (BSSE) from the same institution. As a student, her efforts have earned awards from different national and international software and programming competitions, project show casings as well as publications in various international conferences. She has the experiences of working both in industry and academia. Her core areas of interest are software testing, design, reengineering and refactoring.

Iftekhar Ahmed is a Master's student of Media Informatics at RWTH Aachen University. He pursued his Bachelor of Science in Software Engineering (BSSE) from the Institute of Information Technology, University of Dhaka. He has over 5 years' experience in professional software development. His research interests include Empirical Research in Software Engineering and NLP.

Kishan Kumar Ganguly received the Bachelor of Science in Software Engineering (BSSE) and Master of Science in Software Engineering (MSSE) degrees from the Institute of Information Technology, University of Dhaka. He is currently working as a lecturer in the Institute of Information Technology, University of Dhaka. His research interest includes applications of machine learning and software engineering for self-adaptive system.

S. M. Arif Raian is currently working as Assistant Superintendent of Police (ASP) of Intelligence Wing at Bangladesh Police, Bangladesh. He received his Bachelor of Science in Software Engineering (BSSE) degree from the Institute of Information Technology, University of Dhaka. Before joining at Bangladesh Police, as a software engineer he has worked in professional software development at distinct software companies. He has keen interest in research based innovation and technology. His research engrossment lies in the area of cyber security, cybercrime and digital forensic.

Abdus Satter is a Lecturer at the Institute of Information Technology (IIT), University of Dhaka, Bangladesh. He pursued his Master of Science in Software Engineering (MSSE) and Bachelor of Science in Software Engineering (BSSE) from the same institution with the top score in his class. His core areas of interest are software repository mining, software engineering, web technologies, systems and security. He has numerous awards in various national and international software and programming competitions, hackathons project showcasing.

How to cite this paper: Atish Kumar Dipongkor, Rayhanul Islam, Nadia Nahar, Iftekhar Ahmed, Kishan Kumar Ganguly, S.M. Arif Raian, Abdus Satter, " Reduction of Multiple Move Method Suggestions Using Total Call-Frequencies of Distinct Entities", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.12, No.4, pp. 21-29, 2020. DOI: 10.5815/ijieeb.2020.04.03

