IEICE TRANSACTIONS

on Information and Systems

VOL. E98-D NO. 11
NOVEMBER 2015

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. INF. & SYST., VOL.E98–D, NO.11 NOVEMBER 2015

LETTER

Improvement in Method Verb Recommendation Technique Using Association Rule Mining

Yuki KASHIWABARA[†a)], ***Nonmember***, **Takashi ISHIO**[†], ***Member***, ***and*** **Katsuro INOUE**[†], ***Fellow***

SUMMARY In a previous study, we proposed a technique to recommend candidate verbs for a method name so that developers can consistently use various verbs. In this study, we improve the rule extraction technique proposed in this previous study. Moreover, we confirm that the rank of each correct verb recommended by the new technique is higher than that by the previous technique.

key words: *recommendation, verb in method name, association rule*

1. Introduction

Developers take a considerably long time to understand a program if identifiers poorly represent their roles in the program [1]. Method names are important identifiers for program readability because they are used for understanding the behavior of methods without reading the program. Several techniques are used to help developers to find a better name for a method [2], [3]. Karlsen *et al.* [4] implemented a tool that accurately points out inappropriate verbs used for a method and recommends more appropriate verbs for renaming the method. Yu *et al.* proposed a technique to recommend a verb for a method name using machine learning for automatic naming [3]. In our previous study, we proposed a technique to recommend candidate verbs for a method name so that developers can find a better verb for a method [5]. We extracted the relationship between verbs used in method names and identifiers used in method bodies from existing source files by using association rule mining [6]. We assumed that the behavior of a method is often characterized by identifiers such as method calls and field access in the method body. By using the extracted rules, we recommended candidate verbs likely to be used as a part of a method name, along with the reason for recommendation: *e.g.*, *if a method calls* `next`, `hasNext`, `iterator`, *and* `equals`, *then* `find` *is likely to be a verb representing the behavior*.

In this study, we improve the rule extraction technique proposed in this previous study. We split an identifier into words because it is difficult to extract rules for long identifiers comprising several words. In addition, we use new clues to characterize the usage of a verb and a new threshold to extract rules. To evaluate the improvement relative to the previous technique, we extracted rules from the same training set using both the new and the previous technique and applied them to the same projects. We compared the rank of correct verbs in each list. As a result, we confirm that the ranks of correct verbs recommended by the new technique are, on average, higher than those by the previous technique.

Manuscript received March 23, 2015.
Manuscript revised July 6, 2015.
Manuscript publicized August 13, 2015.

[†]The authors are with Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: k-yuki@ist.osaka-u.ac.jp

DOI: 10.1587/transinf.2015EDL8069

2. Improved Verb Recommendation

Our approach recommends candidate verbs for a method name using association rule mining. This approach consists of two steps. The first step extracts naming association rules from verbs used in method names and the identifiers in method bodies. The second step applies the rules to recommend verbs for a method name. We improved only the first step.

2.1 Extraction of Naming Association Rules

In this step, our approach extracts rules to recommend verb candidates from a training data set using association rule mining. A training data set represents a set of existing projects. Toward this end, our approach takes methods from a training data set, translates the methods to transactions, and extracts rules from the transactions.

First, our approach takes methods from binary files in the training data set. The identifiers used in these methods are more easily extracted from binaries than from source files. We extract transactions from compilable projects. In the implementation, we use the Java byte code framework ASM[*] to extract methods from binary files. We ignore methods compiled without debug information because we extract argument names from debug information in binaries.

Second, our approach generates a set of transactions from methods by translating each method into a transaction that is a set of elements. Each element is represented as pairs of an identifier and its category, like "*category*:*name*," where *category* denotes the category of the identifier and *name* represents the text of the identifier. In the new technique, we extract the following six categories of elements in a target method.

method-verb: A verb used in the name of the target method. As with the previous technique, we do not use stemming. We analyze similar verbs including synonyms individually.

[*]http://asm.ow2.org/

return-type: Return type of the target method.
field-type: Type of field accessed in the target method.
argument-type: Type of argument of the target method.
call-verb: A verb used in the name of a method directly called by the target method.
word: Each word in the name of an argument and in the name of a field that is accessed in the target method and each word in the name of a method directly called by the target method. All words are treated as lowercase. We filter out single-character words.

We add a field type because it represents the external data used by a method as being similar to the argument-type and return-type. We employed identifier splitting to extract rules because a common word among identifiers may represent functional commonalities [7], [8].

Figure 1 shows how a source file is translated into transactions. The source file in Fig. 1 (a) includes a method: `containsName`. Figures 1 (b) and (c) respectively show transactions of *t*(`containsName`) extracted using the new and the previous techniques. *symbols represent elements that are different from one another.

Finally, our approach applies association rule mining to the transactions. In our previous study, we established four conditions.

(1) The antecedent of a rule contains no method-verbs.
(2) The consequent of a rule contains only one method-verb.
(3) The number of items in the antecedent is less than or equal to 4.
(4) The support value is more than or equal to 100.

In addition to these four conditions, we established two conditions.

(5) Lift value is more than or equal to 1.

```
public class NameList implements Serializable {
  String[] nameArray;
  int size;

  public boolean containsName(String n){
    for(int i=0; i<size; i++){
      if(nameArray[i].equals(n)){
        return true;
      }
    }
    return false;
  }
}
```

(a) source code

method-verb:contains
return-type:boolean
argument-type:String
* field-type:String[]
* field-type:int
* word:name
* word:array
* word:size
* call-verb:equals

(b) a transaction of `containsName` method extracted by the new technique

method-verb:contains
return-type:boolean
argument-type:String
* argument-name:n
* field-name:nameArray
* field-name:size
* call:equals

(c) a transaction of `containsName` method extracted by the previous technique

Fig. 1 Example of two transactions.

(6) The item in the antecedent is **not** { return-type:void }.

The lift value represents the performance of the extracted rule. We established conditions (5) and (6) to extract meaningful rules.

2.2 Recommendation of Verbs

This step is the same as in the previous study. In this step, we use rules extracted from a training data set to recommend verbs for a given method. We extract a transaction from the given method and select rules whose antecedents are subsets of the transaction of the given method. If more than one rule recommends the same verb, we use only one rule with the highest confidence value. A list of verbs sorted by descending order of their confidence values is recommended to developers.

3. Evaluation

We compared the accuracy of the new technique with that of our previous technique. We extract two rule sets using the previous and the new techniques, respectively, and we applied each extracted rule set to the same methods in four OSS projects. We regarded **a verb already used in a target method name** as **the correct verb for the target method**. This assumption is the same as in [3], [5]. We have evaluated the accuracy of our approach to answer the following research question: "Can the new technique recommend more correct verbs than the previous technique?" We extracted rules from binary files in 112 OSS projects obtained in Qualitas Corpus (ver.20130901)[†]. 299,482 and 2,743,605 rules are extracted using the previous and the new techniques, respectively, from 1,180,714 methods.

We applied the extracted rules to the training data set itself and four open source projects: BlueJ, NeoDatis, Saxon-HE, and Order Portal used in [9]. The 112 projects mentioned above do not include these four projects.

Table 1 shows an overview of the training data set and the target projects with the results. The domain represents the type of domain of a project. The classification was referred to from [9]. #Method indicates the number of methods in the project, and #Target represents the number of considered methods as described in Sect. 2.1. #Recom and Top5 represent the results. #Recom denotes the number of methods whose existing verbs are recommended in each list (and percentage of methods to the target methods). Top5 indicates the number of methods recommended in the top 5 of each list (and the percentage). (P) and (N) represent the result of the previous and the new techniques, respectively.

To answer the research question, we evaluated whether the verbs currently used in projects can be recommended by the rules. In Fig. 2, the vertical axis represents the percentage of the number of methods for which the existing verbs are recommended. The horizontal axis represents the rank of existing verbs. From Fig. 2, if we check the top 30 of the

[†]http://qualitascorpus.com/

Table 1 Overview and result of training data set and four open source projects.

project	domain	#Method	#Target	#Recom(P)	Top5(P)	#Recom(N)	Top5(N)
Qualitas Corpus	(NONE)	4,671,951	1,180,714	1,064,748(90.2%)	697,017(59.0%)	1,043,437(88.4%)	782,012(66.2%)
BlueJ 3.1.4[†]	desktop application	8,188	2,800	2,394(85.5%)	1,093(39.0%)	2,184(78.0%)	1,351(48.3%)
OrderPortal 10.05.01[††]	web application	41,241	3,392	3,118(91.9%)	1,055(31.1%)	2,812(82.9%)	1,358(40.0%)
Saxon-HE 9.6.0.4[†††]	xml	16,701	6,226	4,784(76.8%)	2,651(42.6%)	5,026(80.7%)	3,220(51.7%)
NeoDatis 1.9.30.689[††††]	database	4,472	1,445	1,224(84.7%)	701(48.5%)	1,230(85.1%)	786(54.4%)
sum	(NONE)	70,602	13,863	11,520(83.1%)	5,500(39.7%)	11,252(81.2%)	6,715(48.4%)

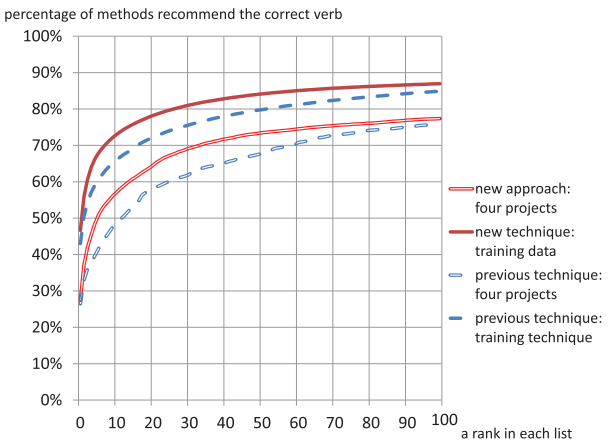

Fig. 2 Rank of correct verbs for methods in four projects and training data set.

recommended list for a method in the training date set using the new technique, we can find existing verbs for 80% of the target methods.

As a result,when using the previous technique, existing verbs are recommended in the top 5 for 39.7% of the methods in four projects and 59.0% of the methods in the training dataset. When using a new technique, existing verbs are recommended for 48.4% of the methods in four projects and 66.2% of the methods in the training dataset. By using Mann-Whitney U test, we confirm that the new technique can recommend correct verbs with a higher rank than the previous technique. The p value $p = 2.2 \times 10^{-16} < .05$ was considered significant. The difference in the median is 3.5. In other words, the new technique recommended correct verbs with 3.5 higher rank, on average, than the previous technique.

Both the field types and the identifier splitting employed by the new technique contributed to this improvement. Introducing field types provides new clues to find commonalities missed by the previous technique. For example, the new technique extracted a rule that some `resolve` methods use a `ClassLoader` object stored in a field. Identifier splitting is also effective because a method name usually includes a verb and additional words. For example, `remove` methods often call other `remove` methods including additional words in their names (e.g., `removeSomething`). The new technique uses such "same action" methods [8].

While the new technique results in a better list of verbs, #Recom(N) is lower than #Recom(P); *i.e.*, the new technique can recommend verbs for less number of methods than the previous technique. The rules extracted by the previous and new techniques can recommend 432 and 405 verbs, respectively (14.1% and 13.2% of 3,060 verbs that appeared in the training data set). The differences are caused by the two conditions added to exclude meaningless rules. For example, the previous technique extracted the following rule for 29 `reload` methods: *if a return of a method is* `void`, *then* `reload` *is likely to be a verb representing the behavior*. Although the rule accidentally recommended the correct verb for the methods, the new technique excluded the rule because methods returning `void` are omnipresent in projects. This is a shortcoming in order to exclude meaningless rules to improve a recommendation list. We believe that this is acceptable because developers find it difficult to choose a verb recommended by such meaningless rules.

4. Threats to Validity

We extracted rules from binary files in open source projects. As several copies of a library are included in different projects, the methods in such a library are learned multiple times. We use the same binary files for learning to compare two techniques, and therefore, we consider that there is low impact.

5. Conclusion

In this study, we proposed a technique to recommend candidates of appropriate method verbs to developers by using naming association rules. We confirmed that the new technique recommended correct verbs with 3.5 higher rank, on average, than the previous technique.

In future study, we need to reduce unnecessary rules. Moreover, we would like to investigate how many verbs developers can check through a subjective experiment.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 25220003 and 26280021.

[†]http://www.bluej.org/
[††]https://launchpad.net/orderportal
[†††]http://saxon.sourceforge.net/
[††††]http://sourceforge.net/projects/neodatis-odb/

References

[1] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? A study of identifiers," 14th IEEE International Conference on Program

Comprehension (ICPC '06), pp.3–12, 2006.
[2] E.W. Høst and B.M. Østvold, "Debugging method names," ECOOP 2009 — Object-Oriented Programming, Lecture Notes in Computer Science, vol.5653, pp.294–317, Springer, Berlin, Heidelberg, 2009.
[3] S. Yu, R. Zhang, and J. Guan, "Properly and automatically naming Java methods: A machine learning based approach," Advanced Data Mining and Applications, Lecture Notes in Computer Science, vol.7713, pp.235–246, Springer, Berlin, Heidelberg, 2012.
[4] E.K. Karlsen, E.W. Høst, and B.M. Østvold, "Finding and fixing Java naming bugs with the Lancelot Eclipse plugin," Proc. ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation, PEPM '12, pp.35–38, 2012.
[5] Y. Kashiwabara, Y. Onizuka, T. Ishio, Y. Hayase, T. Yamamoto, and K. Inoue, "Recommending verbs for rename method using association rule mining," 2014 Software Evolution Week, IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), pp.323–327, 2014.
[6] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," ACM SIGMOD Record, vol.22, no.2, pp.207–216, 1993.
[7] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," 2009 IEEE 31st International Conference on Software Engineering, pp.232–242, 2009.
[8] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," Proc. IEEE/ACM International Conference on Automated Software Engineering, ASE '10, pp.43–52, 2010.
[9] Y. Hayase, Y. Kashima, Y. Manabe, and K. Inoue, "Building domain specific dictionaries of verb-object relation from source code," 2011 15th European Conference on Software Maintenance and Reengineering, pp.93–100, 2011.