Assessing the Differences of Clone Detection Methods Used in the Fault-prone Module Prediction

Masateru Tsunoda
Kindai University
Higashiosaka, Japan
tsunoda@info.kindai.ac.jp

Yasutaka Kamei
Kyushu University
Fukuoka, Japan
kamei@ait.kyushu-u.ac.jp

Atsushi Sawada
Nanzan University
Nagoya, Japan
sawada@se.nanzan-u.ac.jp

***Abstract*—We have investigated through several experiments the differences in the fault-prone module prediction accuracy caused by the differences in the constituent code clone metrics of the prediction model. In the previous studies, they use one or more code clone metrics as independent variables to build an accurate prediction model. While they often use the clone detection method proposed by Kamiya et al. to calculate these metrics, the effect of the detection method on the prediction accuracy is not clear. In the experiment, we built prediction models using a dataset collected from an open source software project. The result suggests that the prediction accuracy is improved, when clone metrics derived from the various clone detection tool are used.**

Keywords— Clone metrics; fault-prone module; prediction performance

I. INTRODUCTION

To enhance software reliability, it is important to find and remove faults in the testing phase. Predicting fault-prone modules is recognized to be a promising approach to finding software faults, and many fault-prone module prediction models are proposed in the software engineering field. When building the prediction model, selecting proper independent variables is important to enhance prediction accuracy.

When selecting independent variables of a fault-prone prediction model, we investigate each software metric such as a complexity metric of the source code from a viewpoint of the relevance to software faults. Existing studies suggested that code clone related metrics are some of the effective alternatives [4], and hence using the clone metrics (e.g., the ratio of the lines of duplicated code per the lines of code) is expected to enhance the performance of fault prediction models. In the literature [2], they tried to use clone metrics as independent variables of the prediction model, to enhance the performance of the model.

Many code clone detection methods have been proposed [4], since code clones affect software maintainability. Each resulting set of code clones differs from another if we use different detection methods or if we use different configurations of one detection method [5]. Relevant code clone metric also differs from one to another. For instance, if we set a smaller value to the 'minimum size of detected clone' parameter, we will get a larger value of the 'ratio of clones to lines of code' metric because of the increasing number of detected clones. Since we focus on the effects caused by different clone metrics used in a fault-prone module prediction model, we will take the differences in method configurations to be same as the differences in detection methods.

Existing studies use clone metrics [2] calculated using the code clones resulting from the detection method proposed by Kamiya et al. [3]. On the other hand, changes in the prediction accuracy are not clear if we alter the prediction model so that it uses another method for code clone detection. Also, the changes in prediction accuracy is not clear when the model uses various clone metrics which is derived from different detection methods simultaneously. In this paper, we assess the differences in the prediction power through several experiments in which we use the model comprised of various clone metrics calculated from various clone detection methods.

II. EXPERIMENT

Overview: To assess the clone metrics the effect of clone metrics which are derived from different detection methods, we have built several different fault-prone module prediction models using these clone metrics, and compared the prediction accuracy. We used dataset measured on open source software, Lucene 2.4.0. The dataset was collected by D'Ambros et al. [1], and it includes complexity metrics and the number of faults after the release. We measured clone metrics from the source code, and merged it to the dataset. The number of modules was 691, and the number of fault modules was 64.

Clone detection tools and metrics: We used four clone detection tools which are widely used [5]: PMD's Copy/Paste Detector (CPD), CCFinderX, Simian, and NiCad3. CPD and CCFinderX is token based, Simian is text based, and NiCad3 is parser based approach. We made two configuration settings on each tool as shown in Table I. One is to find small size

TABLE I. CLONE DETECTION TOOLS AND METRICS

Clone metric	Tool	Configuration
c1	CCFinderX	# of minimum tokens is 50
c2	CCFinderX	# of minimum tokens is 100
c3	Simian	# of minimum LOC is 6
c4	Simian	# of minimum LOC is 9, and ignore identifiers
c5	CPD	# of minimum tokens is 100
c6	CPD	# of minimum tokens is 200, and ignore literals
c7	NiCad3	block clone, and type 2 (renamed) clone
c8	NiCad3	function clone, and type 3-2c (near-miss renamed consistently) clone

clones, and the other type is to find large size clones.

Based on the outputs from these tools, we have calculated the values of clone metrics c1...c8, which are either the number of duplicated lines of code divided by total number of lines of code (total NLOC), or the number of duplicated tokens divided by total NLOC. Table 2 shows Spearman's correlation matrix of the clone metrics. There was not very high correlation (The maximum value was 0.74).

Variety of models: We have built eight prediction models M1…M8 using each metric c1…c8 respectively. In addition to clone metrics, we used 17 variables as candidates of independent variables. They are CK metrics and other object oriented metrics. We have also built the model M9 which includes all of c1…c8 metrics. In order to make a comparison with the effects caused by conventional metrics such as CK metrics, we have built the model M10 which includes none of c1…c8 metrics. Existing studies such as [2] which have adopted clone metrics in prediction models show the improvement of prediction accuracy is limited. We have built M9 since we assume when various detection methods are applied, code clone is characterized from various points of view.

Building and evaluating prediction models: In fault-prone module prediction, a model discriminates whether each module will have faults or not after software release (i.e., when the number of faults after the release is more than zero, the module has faults). To build the models, we used the logistic regression, which is widely used in fault-prone module prediction.

Candidates of independent variables were selected based on Akaike's Information Criterion (AIC). When correlation between independent variables was very high, one of the variables was removed from the candidates. In model M1 ... M9, clone metrics were not removed from the model. Note that metric c6 was removed from M9 because of high correlation.

To evaluate the models, we used the area under the curve (AUC). AUC is the more appropriate criterion for discriminant methods than other criteria like F1 score. The value range of AUC is [0, 1], and higher AUC means that prediction accuracy of the method is high. To calculate the evaluation criterion, we applied 5-fold cross validation (In each subset, the rate of fault-prone modules was same), and repeated it two times to increase the number of evaluations.

Result: The result is shown in Table 3. We statistically tested the difference from M10, using the Wilcoxon signed-rank test at a significance level of 0.05. In the table, bolded rows indicates there was statistical difference. Focusing on M1 … M8, M1 and M3 had statistical difference, although the difference is small (0.03). On configurations of c1 and c2, minimum length of clone is small. So, in this dataset, it may be good that the minimum length of clone is small. So, choosing proper detection tools and configurations to derive clone metrics would not be ignorable to improve prediction accuracy.

Compared with M1 … M8, M9 had relatively large difference from M10 (0.05), and it was statistically different. So, using the sole detection tool is not very effective, but using various tool and configurations to derive clone metrics would be effective, compared with using only conventional metrics.

TABLE II. CORRELATION MATRIX OF THE CLONE METRICS

Metric	c1	c2	c3	c4	c5	c6	c7	c8
c1	1	0.74	0.47	0.51	0.52	0.67	0.65	0.34
c2	0.74	1	0.54	0.55	0.41	0.55	0.60	0.30
c3	0.47	0.54	1	0.62	0.24	0.45	0.48	0.25
c4	0.51	0.55	0.62	1	0.28	0.50	0.52	0.29
c5	0.52	0.41	0.24	0.28	1	0.57	0.48	0.10
c6	0.67	0.55	0.45	0.50	0.57	1	0.62	0.26
c7	0.65	0.60	0.48	0.52	0.48	0.62	1	0.34
c8	0.34	0.30	0.25	0.29	0.10	0.26	0.34	1

TABLE III. PREDICTION ACCURACY OF THE MODELS

Model	Clone metrics	AUC	Difference from M10	p-value
M1	**c1**	**0.69**	**0.03**	**0.02**
M2	c2	0.67	0.01	0.24
M3	**c3**	**0.69**	**0.03**	**0.02**
M4	c4	0.66	0.01	0.53
M5	c5	0.67	0.01	0.08
M6	c6	0.65	0.00	0.88
M7	c7	0.67	0.01	0.37
M8	c8	0.66	0.00	0.75
M9	**c1...c8**	**0.71**	**0.05**	**0.02**
M10	-	0.66	-	

III. CONCLUSIONS

In this paper, we compared the prediction accuracy of fault-prone module prediction models, when using clone metrics derived from the various clone detection tool and configurations. The experimental result showed using various clone metrics would be effective, compared with using only conventional source metrics such as CK metrics.

ACKNOWLEDGMENT

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (C) (No. 25330090)], and by Nanzan Univ. Pache Research Subsidy I-A (2015).

REFERENCES

[1] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," Proc. international working conference on mining software repositories (MSR), pp.31-41, 2010.

[2] Y. Kamei, H. Sato, A. Monden, S. Kawaguchi, H. Uwano, M. Nagura, K. Matsumoto, and N. Ubayashi, "An Empirical Study of Fault Prediction with Code Clone Metrics," Proc. Joint Conf. of the Int. Workshop on Software Measurement and Int. Conf. on Software Process and Product Measurement (IWSM-MENSURA) , pp.55-61, 2011.

[3] T. Kamiya, S., Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," IEEE Transactions on Software Engineering, vol.28, no.7, pp.654-670, 2002.

[4] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," Information and Software Technology, vol.55, no.7, pp.1165-1199, 2003.

[5] T. Wang, M. Harman, Y. Jia, and J. Krinke, "Searching for better configurations: a rigorous approach to clone evaluation," Proc. joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering (ESEC-FSE), pp.455-465, 2013.