2014 6th International Workshop on Empirical Software Engineering in Practice

A Cross-Project Evaluation of Text-based Fault-prone Module Prediction

Osamu Mizuno, Yukinao Hirata
Graduate School of Science and Technology, Kyoto Institute of Technology, Kyoto, Japan
E-mail: o-mizuno@kit.ac.jp

***Abstract*—In the software development, defects affect quality and cost in an adverse way. Therefore, various studies have been proposed defect prediction techniques. Most of current defect prediction approaches use past project data for building prediction models. That is, these approaches are difficult to apply new development projects without past data. In this study, we focus on the cross project prediction that can predict faults of target projects by using other projects. We use 28 versions of 8 projects to conduct experiments of the cross project prediction and intra-project prediction using the fault-prone filtering technique. Fault-prone filtering is a method that predicts faults using tokens from source code modules. Additionally, we try to find an appropriate prediction model in the fault-prone filtering, since there are several ways to calculate probabilities. From the results of experiments, we show that using tokens extracted from all parts of modules is the best way to predict faults and using tokens extracted from code part of modules shows better precision. We also show that the results of the cross project predictions have better recall than the results of the intra-project predictions.**

I. Introduction

In software development defects affect quality and cost in an adverse way. Therefore, various studies have been proposed the defect prediction techniques. Most of current studies that predict defects use past project data for building a prediction model. These prediction models can capture characteristics of projects. However, if there is no data about past projects, it is difficult to use these approaches. That is, these approaches are difficult to apply new development projects. There are studies [1]–[4] for solving such a problem. These studies have been aimed to predict defects in a project using other project data. We call such types of prediction using other project as the cross project prediction. The predictive ability of the cross project prediction depends on characteristics between training project and target project (i.e. similar characteristics lead good result). For this reason, the study [5] has been conducted to perform clustering on software projects in order to identify groups of software projects with similar characteristic. Rahman et al. reported that cross-project prediction performance is no worse than within-project performance, and substantially better than random prediction [6].

These studies often use basic software metrics like the line of code (LOC), cyclomatic complexity, CK object oriented metrics [7] and Code Churn [8]. However, we use a method called fault-prone filtering based on text filtering for predicting defects in this study. This approach uses tokens extracted from modules as a metric for prediction. We can extract these tokens from kind of part in modules (e.g. code and comment). In addition, there are no steady way of selecting filter. Therefore, this approach needs to select a kind of tokens and a filter that we use to predict as well as other approaches select metrics and learning algorithms. In addition, the ability of our approach depend on tokens contained in modules. Hence, if we conduct fault-prone filtering in cross project prediction situation, it is a legitimate question whether we can predict faults.

We conduct experiments using 28 versions (8 projects) and investigate related to the following question:

- Can we conduct the cross project prediction using fault-prone filtering?
  Tokens are most important in the fault-prone filtering. Our approach needs tokens that are contained in both training and target projects to conduct cross project prediction.

Main findings of this study is summarized as follow:

- Intra-project prediction show better precision than cross project prediction. In contrast, recall is better in cross project prediction.

The rest of this paper is structured as follows: In Section II, we describe fault-prone filtering. In Section III, we explain target projects and evaluation measures. In Section IV, we describe experiments and show results of predictions related to the cross project prediction. In Section V, we describe related works. Conclusions are given in Section VI.

II. Fault-prone filtering

A. Basic Idea

The basic idea of fault-prone filtering [9] is inspired by the spam mail filtering. In the spam e-mail filtering, the spam filter first trains both spam and ham e-mail messages from the training data set. Then, an incoming e-mail is classified into either ham or spam by the spam filter.

This framework is based on the fact that spam e-mail usually includes particular patterns of words or sentences. From the viewpoint of source code, a similar situation usually occurs in faulty software modules. That is, similar faults may occur in similar contexts. We assumed that faulty software modules have similar patterns of words or sentences as spam e-mail messages have. To obtain such features, we adopted the spam filter in fault-prone module prediction.

978-1-4799-6666-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IWESEP.2014.9

Intuitively speaking, we try to introduce a new metric as a fault-prone predictor. The metric is "frequency of particular words". In detail, we do not treat a single word, but use combinations of words for the prediction. Thus, the frequency of a certain length of words is the only metric used in our approach.

We then try to apply a spam filter to identify fault-prone modules. We named this approach as "fault-prone filtering". That is, a learner first trains both faulty and non-faulty modules. Then, a new module can be classified into either fault-prone or not-fault-prone using a classifier.

B. Extraction of Tokens

We conduct experiments using the fault-prone filtering in this study. In these experiments, we use tokens extracted from specific parts of modules (e.g. code and comment). We explain the specific parts of modules and how to extract tokens from each part in this section.

1) Definition of Comment Lines: In order to investigate the ability of fault-prone prediction, we distinguish the contents of source code modules into two classes. Code lines describe a list of operations that developers would like to realize on computers. Comment lines include descriptions of code lines, usage of methods or modules. Code lines are written in a specific programming language, but comment lines are written in a free form.

Original implementation of fault-prone filtering did not distinguish the comment lines and code lines. The source code module is passed into text filter without any modification. However, since code lines and comment lines have different roles in source code modules, we need to consider such difference in the fault-prone filtering.

For example, comments are usually placed near the difficult codes. Therefore, learning the contents of comment lines may be useful to identify the bug-related part in modules. Actually, previous research [10] shows that the prediction using tokens in comments is better than the prediction of tokens in code.

In this study, we treat a java class file as a software module. Developers can write comments anywhere in the module. Those comments are classified into three types by the written form in Java [11]. In this study, we defined the following comment classes:

- TC_{EOL} : End-Of-Line Comments (e.g. `//`)
- TC_{BLK} : Block Comments, Single-Line Comments, Trailing Comments (e.g. `/* .. */`)
- TC_{DOC} : Documentation Comments (e.g. `/** ... */`)

2) Structure of Line: Copied and pasted source code, which is called "code clone", affects to software quality. There are several studies [12], [13] that detect faults using code clone. These studies use structure of code clone to detect faulty code. Code clones are parts of structure in module. In contrast, we use all of structure in modules to predict faults because all structures in a module affect to the trend of faults of the module.

Next, we explain how to extract structure from modules. In order to extract structure, we apply following steps to modules.

Step 1 Remove all types of comments, '{' and '}'.
Step 2 Replace identifiers include numbers, strings with double quotes and character with single quotes into I, S and C, separately.
Step 3 Remove all white spaces and tabs.

Applying by these steps, we get structure of modules. In this study, we use each lines as tokens for fault-prone filtering. We define a class that consist of these tokens as TC_{LINE}.

3) Tokenization: In this study, we define 8 kinds of token class extracted from the source code: TC_{ALL}, TC_{CODE}, TC_{EOL}, $TC_{CODE+EOL}$, TC_{COM}, TC_{BLK}, TC_{DOC} and TC_{LINE}. Here, TC_{CODE} token class includes tokens extracted from code except comments in modules. TC_{ALL} token class includes TC_{CODE} and TC_{COM} token classes. That is, TC_{ALL} token class uses all part of a module. In similar way, $TC_{CODE+EOL}$ token class includes TC_{CODE} and TC_{EOL} token classes. The reason that we use this combination is TC_{EOL} shows that better prediction result in previous study [14].

We conduct to tokenize above all token classes in the same way except TC_{LINE} token class (Tokenization of TC_{LINE} class already explained in section II-B2). We tokenize contents into identifiers, numbers, escape sequence, keywords, operators in Java language.

III. Target Projects and Evaluation Measures

A. Target Projects

In this study, we use 8 projects, Apache Ant, Eclipse, jEdit, Apache Lucene, Apache POI, Apache Velocity, Apache Xalan and Apache Xerces, for our experiments. In order to conduct our experiments, we need to collect faulty information of modules in each project. We adopt faulty data in the PROMISE [15] repository. There are about one hundred data about faults. In our approach, we need information about modules and modules themselves. Therefore, we use data that contain module names in order to satisfy our requirement. Consequently, we select 8 projects described above. All of the projects but Eclipse are donated by Jureczko [5], [16] and Eclipse are donated by Zimmermann [17].

These data sets contain the number of faults for each module. However, our approach determines existence or non-existence of faults. Therefore, we define a faulty module as a module that contains one or more faults and if a module does not contain any faults, we define the module as a non-faulty module. We show the number of faulty modules and non-faulty modules for each version in Table I. As shown in the Table I, we use some versions for each project. The number of modules in this table is fewer than the number of

Table II
Classification result matrix

		Classified	
		non-fault-prone	fault-prone
Actual	non-faulty	True negative (TN)	False positive (FP)
	faulty	False negative (FN)	True positive (TP)

modules in the data set because we only use data about modules that are contained in each release version. This happens, if the data sets include data about modules that were made between releases.

B. Evaluation Measures

Table II shows a classification result matrix. True negative (TN) shows the number of modules that are classified as non-fault-prone, and are actually non-faulty. False positive (FP) shows the number of modules that are classified as fault-prone, but are actually non-faulty. On the contrary, false negative (FN) shows the number of modules that are classified as non-fault-prone, but are actually faulty. Finally, true positive (TP) shows the number of modules that are classified as fault-prone which are actually faulty.

In order to evaluate the results, we prepare three measures: recall, precision, and accuracy. Recall is the ratio of modules correctly classified as fault-prone to the number of entire faulty modules. Recall is defined by Equation (1).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{1}$$

Precision is the ratio of modules correctly classified as fault-prone to the number of entire modules classified fault-prone. Precision is defined by Equation (2).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Accuracy is the ratio of correctly classified modules to the entire modules. Accuracy is defined by Equation (3).

$$\text{Accuracy} = \frac{TP + TN}{TN + TP + FP + FN} \tag{3}$$

Since recall and precision are in the trade-off, F_1-measure is used to combine recall and precision. F_1-measure is defined by Equation (4).

$$F_1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{4}$$

In this definition, recall and precision are evenly weighted.

In addition, we introduce an evaluation measure related to precision. Precision defined by Equation (2) is affected by ratio of faulty modules (see Table I). For example, if ratio of faulty modules of the target version is high, precision also tends to be high in general. In order to make a new evaluation measure that eliminate the effect of ratio of faulty modules, first, we define the ratio of faulty using TN, TP, FP and FN as follows:

$$F_r = \frac{TP + FN}{TN + TP + FP + FN} \tag{5}$$

Next, we define a new evaluation measure through Equation (6) using Precision and F_r. This measure reduces the impact of ratio of faulty modules.

$$PF_r = \text{Precision} - F_r \tag{6}$$

IV. Cross Project Prediction

In this section, we conduct the cross project predictions using fault-prone filtering. In cross project prediction, we use one or more projects as training data to predict other projects. We need at least one version of target project in intra-project prediction. Hence, we can only apply intra-project prediction to the versions developed after the second version. However, we need no older versions of a target project in cross project prediction. Therefore, if we have only one version of a target project, we can apply the cross project prediction to the project.

A. Ex. 1: Prediction Using A Single Project

We consider that cross project prediction is more difficult task than intra-project prediction. In this experiment, we conduct simple condition cross project prediction. That is, a single project is used as training data, in order to investigate whether fault-prone filtering works in cross project prediction.

1) Experimental Method: In Ex. 1, we conduct the cross project prediction using single project as training data. In particular, we use all of versions as training data in a project, and predict the other projects. For example, when we use Ant project as training data, the other projects, Eclipse, jEdit, Lucene, POI, Velocity, Xalan and Xerces, are target data. That is, training and target pairs are (All of Ant versions, Ec20), ..., (All of Ant versions, Ec30), (All of Ant versions, Je32), ..., (All of Ant versions, Xerces1.6). In this experiment, we use TC_{ALL} token class extracted from modules.

2) Results and Discussions: We show the results of prediction in Table III. In this table, if the value of PF_r is equal to 0 or fewer, that is, if the values of precision are better than the value of prediction by random prediction, we indicate by '✓'. The sign '−' means that we do not conduct the combination of prediction because training and target data are the same in these combinations. We find that most projects work well as training data. The results in Table III show that some projects (POI and Velocity) are not appropriate for predicting other projects. Therefore, we excluded these two projects from further experiments.

Table IV shows the average evaluation measures of results in each project. From these results, Eclipse is the best project for predicting the other projects. These results are caused by

Table I
Target versions

abbrev	project	faulty	non-faulty	ratio of faulty	abbrev	project	faulty	non-faulty	ratio of faulty
An14	Ant1.4	40	137	22.6 %	Po15	POI1.5	141	94	60.0 %
An15	Ant1.5	32	260	11.0 %	Po20	POI2.0	37	272	12.0 %
An16	Ant1.6	92	258	26.3 %	Po25	POI2.5	247	132	65.2 %
An17	Ant1.7	166	575	22.4 %	Po30	POI3.0	281	157	64.2 %
Ec20	Eclipse2.0	975	5754	14.5 %	Ve14	Velocity1.4	147	48	75.4 %
Ec21	Eclipse2.1	854	7034	10.8 %	Ve15	Velocity1.5	142	72	66.4 %
Ec30	Eclipse3.0	1568	9025	14.8 %	Ve16	Velocity1.6	78	151	34.1 %
Je32	jEdit3.2	90	170	34.6 %	Xa24	Xalan2.4	110	566	16.3 %
Je40	jEdit4.0	75	218	25.6 %	Xa25	Xalan2.5	387	375	50.8 %
Je41	jEdit4.1	79	221	26.3 %	Xa26	Xalan2.6	411	464	47.0 %
Je42	jEdit4.2	48	307	13.5 %	Xe21	Xerces1.2	71	368	16.2 %
Je43	jEdit4.3	11	476	2.3 %	Xe13	Xerces1.3	69	383	15.3 %
Lu20	Lucene2.0	91	95	48.9 %	Xe14	Xerces1.4	210	118	64.0 %
Lu22	Lucene2.2	143	91	61.1 %					
Lu24	Lucene2.4	203	127	61.5 %					

Table III
Combinations of training and targets with good prediction

	training							
target	An*	Ec*	Je*	Lu*	Po*	Ve*	Xa*	Xe*
An14	-	✓	✓					✓
An15	-	✓	✓	✓			✓	✓
An16	-	✓	✓	✓		✓	✓	✓
An17	-	✓	✓	✓		✓	✓	✓
Ec20	✓	-	✓	✓	✓		✓	✓
Ec21	✓	-	✓		✓		✓	✓
Ec30	✓	-	✓	✓	✓		✓	✓
Je32	✓	✓	-		✓		✓	✓
Je40	✓	✓	-	✓	✓		✓	✓
Je41	✓	✓	-	✓	✓	✓	✓	✓
Je42	✓	✓	-	✓	✓		✓	✓
Je43	✓	✓	-	✓	✓		✓	✓
Lu20	✓	✓	✓	-	✓		✓	✓
Lu22	✓	✓	✓	-	✓	✓	✓	✓
Lu24	✓	✓	✓	-	✓	✓	✓	✓
Po15	✓	✓	✓		-		✓	✓
Po20	✓	✓	✓	✓	-		✓	✓
Po25	✓	✓	✓		-		✓	✓
Po30	✓	✓	✓	✓	-		✓	✓
Ve14	✓	✓		✓	✓	-	✓	
Ve15	✓	✓	✓	✓	✓	-	✓	✓
Ve16	✓	✓	✓	✓	✓	-	✓	✓
Xa24	✓	✓	✓	✓	✓		-	✓
Xa25	✓	✓	✓	✓		✓	-	✓
Xa26	✓	✓	✓	✓		✓	-	✓
Xe12	✓	✓	✓	✓			✓	-
Xe13	✓	✓	✓	✓			✓	-
Xe14	✓	✓	✓	✓			✓	-

Table IV
Average results using a single project in the cross project prediction

	Accuracy	Recall	Precision	F_1
Ant	0.651	0.323	0.598	0.356
Eclipse	0.621	**0.529**	0.555	**0.447**
jEdit	0.636	0.201	**0.647**	0.258
Lucene	0.628	0.233	0.425	0.231
Xalan	**0.671**	0.251	0.596	0.305
Xerces	0.636	0.307	0.615	0.285

the number of modules in Eclipse project. Eclipse is more than ten times larger than the other projects in the viewpoint of the number of modules. Eclipse can adjust and predict various projects because of the enough number of modules.

Table V
Average results using single project and all projects

	Accuracy	Recall	Precision	F_1
Eclipse	0.621	0.529	**0.555**	0.447
All Projects	**0.625**	**0.595**	0.504	**0.471**

B. Ex. 2: Prediction Using All Other Project

In this experiment, we investigate three things in cross project prediction. 1) do the results of prediction using all projects show better than the results using a single project, 2) trends of results using some kinds of token classes, 3) comparison of the results of intra-project prediction and cross project prediction.

1) Experimental Method: In this experiment, we predict a target project using all the other training projects. That is, when we select a version as target data, we use all versions for training except versions that belong to the target project. For example, when we use Ant as a target project, training and target pairs are (All versions except Ant's, An14), (All versions except Ant's, An15), (All versions except Ant's, An16) and (All versions except Ant's, An17). We predict all versions based on above rule about selecting training data. In addition, we conduct the prediction using TC_{ALL}, TC_{CODE}, TC_{COM}, TC_{EOL}, TC_{BLK}, TC_{DOC}, $TC_{CODE+EOL}$ and TC_{LINE} token classes.

2) Results and Discussions: Table V shows the comparison of the results using a single project (Eclipse) and all projects except the target project (this experiment). From this table, we find that the results are improved in the values of recall and F_1. Therefore, we concluded that combining all projects improves the results.

We show the result of the evaluation measures for each token class in Figure 1. From the viewpoint of F_1 value, TC_{ALL}, TC_{COM}, TC_{BLK} and TC_{DOC} show better results in cross project prediction. However, as shown in Table VI, TC_{COM}, TC_{BLK} and TC_{DOC} show low average PF_r. Therefore, predictions using TC_{COM}, TC_{BLK} and TC_{DOC}

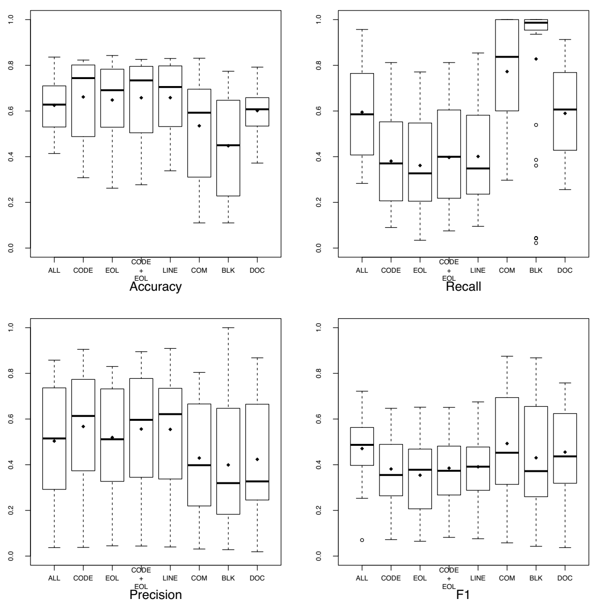

Figure 1. Comparison of results using different tokens in cross project prediction

are impractical. By comparing TC_{ALL}, TC_{CODE}, TC_{EOL}, $TC_{CODE+EOL}$ and TC_{LINE}, obviously, TC_{ALL} shows better F_1 value than TC_{CODE}, TC_{EOL}, $TC_{CODE+EOL}$ and TC_{LINE} in cross project prediction. As a result, using TC_{ALL} token class is the best way of prediction in the cross project prediction. However, if we need better precision, we can use TC_{CODE}, $TC_{CODE+EOL}$ or TC_{LINE}.

Table VIII shows comparison of intra-project and cross project prediction results. Results of intra-project prediction show better precision than results of cross project prediction. In contrast, the results of cross project prediction show better recall. Such trend is also shown in the reference [2].

Table VII shows the number of kinds of tokens extracted from TC_{ALL}. The row of "All Projects" in Table VII related to the cross project prediction. Therefore, tokens are used for predicting faults in two or more projects. Other rows like "Ant Project" are related to the intra-project prediction. Therefore, tokens are used for predicting faults in two or more versions. From Table VII, we find that kinds of tokens are used in the cross project prediction is low (see row of "All Projects"). However, we can predict faults by these tokens (see Figure 1 and Table VIII). Therefore, we can consider that tokens used in several projects are important for predicting faults. In other words, these common tokens between several projects characterize either faulty and non-faulty modules beyond projects. From this result, tokens (structures) extracted from TC_{ALL}, TC_{CODE}, TC_{EOL}, $TC_{CODE+EOL}$ and TC_{LINE} are potentially related to generic fault-proneness.

Table VII
KINDS OF TOKENS EXTRACTED FROM TC_{ALL}

extracted from	in one project	in two or more projects	sum	used in prediction
All Projects	401,903	28,227	430,130	6.6 %
extracted from	**in one version**	**in two or more versions**	**sum**	**used in prediction**
Ant Project	14,447	18,796	33,243	56.5 %
Eclipse Project	114,922	201,746	316,668	63.7 %
jEdit Project	9,229	24,796	34,025	72.9 %
Lucene Project	5,385	9,688	15,073	64.3 %
POI Project	2,778	17,287	20,065	86.2 %
Velocity Project	2,517	8,777	11,294	77.7 %
Xalan Project	10,430	34,282	44,712	76.7 %
Xerces Project	2,040	19,110	211,50	90.4 %

Table VIII
COMPARISON OF INTRA- AND CROSS-PROJECT PREDICTION

Training	Target	Accuracy	Recall	Precision	F_1
An14+An15+An16	An17	**0.725**	0.693	**0.429**	**0.530**
All except An*	An17	0.667	**0.777**	0.381	0.511
Ec20+Ec21	Ec30	**0.841**	0.381	**0.457**	**0.416**
All except Ec*	Ec30	0.826	**0.411**	0.411	0.411
Je32+Je40+Je41+Je42	Je43	**0.764**	0.545	**0.052**	**0.094**
All except Je*	Je43	0.620	**0.636**	0.037	0.070
Lu20+Lu22	Lu24	0.591	0.433	0.815	0.566
All except Lu*	Lu24	**0.630**	**0.478**	**0.858**	**0.614**
Xa24+Xa25	Xa26	0.649	0.275	**0.926**	0.424
All except Xa*	Xa26	**0.728**	**0.752**	0.694	**0.722**
Xe12+Xe13	Xe14	0.409	0.095	**0.833**	0.171
All except Xe*	Xe14	**0.488**	**0.324**	0.723	**0.447**

V. RELATED WORKS

In this section, we describe related works that conducted cross project prediction.

Turhan et al. [2] conducted cross-company and within-company defect predictions using 10 project data from 8 different companies. They showed that cross-company predictions increase probability of detection (*pd*) and decrease probability of false (pf). In order to improve pf value, they proposed an approach that selects training data using k-nearest neighbor. However, the results of within-company is better than the results of cross-company with k-nearest neighbor. Therefore, they concluded that if there are no within-company data, we can use cross-company prediction with k-nearest neighbor and start to collect within-company data. After a few hundred examples are available, it can be switched to use within-company data to predict faults.

Zimmermann et al. [1] conducted 622 cross project prediction and showed that only 21 prediction (3.4%) satisfy their criteria (accuracy, recall and precision are greater than 0.75). They investigate the relation between factors for predicting and the results and conclude that projects in the same domain do not work to build accurate prediction models.

He et al. [4] showed that the results of cross project prediction using suitable training data are better than the results of intra-project prediction. They proposed an approach that selects training data properly. They showed that their approach can find proper training data set for 24 out of 34

Table VI
AVERAGE PF_r VALUES IN CROSS PROJECT PREDICTION

	TC_{ALL}	TC_{CODE}	TC_{EOL}	$TC_{CODE+EOL}$	TC_{LINE}	TC_{COM}	TC_{BLK}	TC_{DOC}
average PF_r	0.152	0.216	0.168	0.205	0.204	0.078	0.048	0.072

versions.

Watanabe et al. conducted inter project prediction using the projects of the same domain (text editor) but different languages (Java and C++) [3]. They proposed an approach called metrics compensation that normalizes metrics between different data sets based on average value of each metric. They showed that the results of inter project prediction are improved by their approach.

VI. CONCLUSIONS

From the results of cross project prediction using kinds of tokens, TC_{ALL}, TC_{CODE}, TC_{EOL}, $TC_{CODE+EOL}$ and TC_{LINE} can use in fault-prone filtering. Especially, TC_{ALL} show the best result in cross project prediction. From the results of prediction using TC_{LINE}, we can say that structures of modules relate to faulty. From the results of cross prediction, there are tokens and structures related to fault-proneness potentially.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 24500038.

REFERENCES

[1] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 91–100.

[2] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Engg.*, vol. 14, pp. 540–578, October 2009.

[3] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, 2008, pp. 19–24.

[4] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, pp. 1–33, 2011.

[5] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–10.

[6] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 61:1–61:11.

[7] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[8] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. of 27th International Conference on Software Engineering*, 2005, pp. 284–292.

[9] O. Mizuno and T. Kikuno, "Training on errors experiment to detect fault-prone software modules by spam filter," in *Proc. of 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, 2007, pp. 405–414.

[10] O. Mizuno and Y. Hirata, "Fault-prone module prediction using contents of comment lines," in *Proc. of International Workshop on Empirical Software Engineering in Practice 2010 (IWESEP2010)*, 12 2010, pp. 39–44, NAIST, Nara, Japan.

[11] S. Microsystems, "5-comments," http://java.sun.com/docs/codeconv/html/CodeConventions.doc4.html, Accessed Feb 3, 2012.

[12] K. Sawa, Y. Higo, and S. Kusumoto, "Proposal and evaluation of an approach to find bugs using difference information of code clone detection tools," *Technical report of IEICE. SS*, vol. 108, no. 173, pp. 67–72, 2008-07-24, (In Japanese).

[13] S. Morisaki, N. Yoshida, Y. Higo, S. Kusumoto, K. Inoue, K. Sasaki, K. Murakami, and K. Matsui, "Empirical evaluation of similar defect detection by code clone search," *The IEICE Trans. on information and systems (Japanese edition)*, vol. 91, no. 10, pp. 2466–2477, 2008-10-01, (In Japanese).

[14] Y. Hirata and O. Mizuno, "Investigating effects of tokens on detecting fault-prone modules by text filtering," in *Proc. of 22nd International Symposium on Software Reliability Engineering (ISSRE2011), Supplemental proceedings*, no. 3-2, 11 2011, hiroshima, Japan.

[15] G. Boetticher, T. Menzies, and T. Ostrand, *PROMISE Repository of empirical software engineering data repository*, http://promisedata.org/, West Virginia University, Department of Computer Science, 2007. [Online]. Available: http://promisedata.org/

[16] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," in *Models and Methodology of System Dependability. Proc. of RELCOMEX 2010: Fifth International Conference on Dependability of Computer Systems DepCoS*, ser. Monographs of System Dependability, Wrocław, Poland, 2010, pp. 69–81.

[17] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proc. of the Third International Workshop on Predictor Models in Software Engineering*, May 2007, p. 9.