Adapting a Fault Prediction Model to Allow Inter Language Reuse

Shinya Watanabe
Faculty of Engineering
Shinshu University
4-17-1 Wakasato
Nagano 380-0935, Japan
t045094@shinshu-u.ac.jp

Haruhiko Kaiya
Faculty of Engineering
Shinshu University
4-17-1 Wakasato
Nagano 380-0935, Japan
kaiya@acm.org

Kenji Kaijiri
Faculty of Engineering
Shinshu University
4-17-1 Wakasato
Nagano 380-0935, Japan
kaijiri@cs.shinshu-u.ac.jp

ABSTRACT

An important step in predicting error prone modules in a project is to construct the prediction model by using training data of that project, but the resulting prediction model depends on the training data. Therefore it is difficult to apply the model to other projects. The training data consists of metrics data and bug data, and these data should be prepared for each project. Metrics data can be computed by using metric tools, but it is not so easy to collect bug data. In this paper, we try to reuse the generated prediction model. By using the metrics and bug data which are computed from C++ and Java projects, we have evaluated the possibility of applying the prediction model, which is generated based on one project, to other projects, and have proposed compensation techniques for applying to other projects. We showed the evaluation result based on open source projects.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, product metrics*

General Terms

Measurement, Experimentation

Keywords

error prone, metrics, inter language prediction, open source

1. INTRODUCTION

Various quality prediction methods based on metrics, especially fault prediction methods, are proposed and their comparison is reported [22, 8], especially fault prediction methods, but most of these methods require learning, especially if good prediction is needed. They predict some quality factor by using parameter and/or decision trees, which are generated with learning process, so training data is needed and each prediction model depends on that training data, that is, on used domain, environment, and languages. As a result, it is difficult to reuse these models for other domain, environment, or languages. For example, COCOMO [9], the representative effort prediction model, proposed effort prediction models, but it requires several customizing factors, and these factors may be obtained from the past similar and/or the same projects. As a result, it is difficult to predict about a new software project, and in ordinary case prediction will be done for a new release based on the past release of the same project. Several researches [14, 13, 16, 15, 18, 19] considered intra project and/or cross company prediction, but the prediction result is not so good comparing with inter project prediction. In almost researches the prediction scheme is reused as-is and no adaptation is considered.

If some adaptation of prediction schemes based on metrics is applied, it becomes possible to apply the prediction model acquired from one project to another project and accuracy will be improved. Such adaptation will use only metric data, and these data can be computed automatically by using some metric tools. For the case of effort prediction, some general factors are proposed, but for quality prediction, no such factors and/or adaptation methods are proposed. Previous researches [23] only show the possibility of as-is intra project and cross company reuse.

In this paper, we consider the inter language model reuse for error proneness prediction. We show the ordinal prediction flow in Figure 1. In error proneness prediction, we use the pair of source code metric data and bug data as training data. Test data is metric data of the project whose error proneness is to be predicted. Training data is derived from project A and test data from project B. In intra project prediction, project B is a succeeding release of project A, but in inter project prediction, project A and project B are releases of different projects. Source code metric data can be computed automatically by using metric tools, but bug data need to be computed based on data of version control systems and/or bug tracking systems, and it is costly and the result is imprecise, so if reuse based on the training data of other projects becomes possible, it becomes easier to predict error prone modules.

Our hypotheses to be researched are as follows:

- ✓ **Hypothesis 1**: The relation between metrics and quality is different from each other in languages.

The sentence "It is possible to achieve inter-project reuse of error-proneness prediction models" means that prediction models developed to identify error-prone areas in one software project can be effectively used in other different software projects. In other words, the techniques and data used to predict where errors are likely to occur in one project can be transferred and applied to predict error-prone areas in other projects, rather than having to create a new prediction model from scratch for each new project. This implies that the prediction models have a level of generality and adaptability that allows them to be useful across various projects, potentially saving time and resources in software development and maintenance.

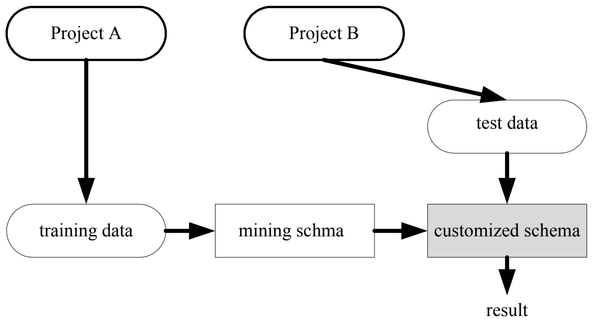

Figure 1: Error Prediction By Using Mining Schema

- **Hypothesis 2**: It is possible to achieve inter project reuse of error proneness prediction models.

We have evaluated the above hypotheses by using open source projects. For this purpose, we have selected two projects from sourceforge.net [5] whose domain and size are similar, but languages used are different.

In Chapter 2, we describe about the characteristics of the target projects and the sampling method, and in Chapter 3 we show the intra project prediction result. In Chapter 4 we show the inter project prediction result and the proposed compensation method, and validate hypothesis 1 and 2. We show the novelty of our method by considering related researches in Chapter 5. In Chapter 6, we evaluate our result and describe the availability of our compensation method and future problems.

2. MINING SCHEMA AND OBJECTIVE APPLICATIONS

In order to validate the above hypotheses we need at least two projects whose target domain and size are similar but only the languages are different, so we selected from sourceforge.net [5] the following two projects and three releases for each project:

- Sakura Editor [3]
  1. sakura_R1.3.0.0 (2002)
  2. sakura_R1.5.0.0 (2005)
  3. sakura_R1.6.2.0 (2007)
- JEdit [1]
  1. jEdit_4-0-final (2002)
  2. jEdit_4-2-final (2004)
  3. jEdit_4-3-pre12 (2007)

The target domain is text editor. We show the characteristics of each project in Table 1. The LOC and the number of files are based on the past releases (sakura_R1.3.0.0, jEdit_4-0-final). There is large difference in the number of developers, and the number of files of jEdit is twice as many as that of Sakura Editor(249 v.s. 538), but LOC is similar (131,275 v.s. 169,107), so we have selected these two projects as our target.

Our objective is to predict the error prone files based on metric data, so we need metric data and bug data. Source code can be obtained from sourceforge.net. The metric data needs to be computed based on the same method, so we used the metric tool family, Understand for C and Understand for Java [4]. These tools compute 63 metrics and we selected the following eight metrics including C&K object oriented metrics [20].

- WMC(Weighted Methods per Class)
- DIT(Depth Inheritance Tree)
- NOC(Number Of Children)
- CBO(Coupling Between Object Classes)
- RFC(Response For Class)
- LCOM(Lack of Cohesion Metric)
- NPM(Number of Public Methods)
- LOC(Line Of Code)

We computed the number of bugs by using SVN repositories of each project as follows:

1. We identified the past release point, that is release in 2002 for each project.
2. We got the log data from that point to current (2007) for each file (java file or C++ file) in the projects by using log subcommand of svn command. We excluded header files in C++.
3. We searched the word (bug or fixed) and counted the frequency of appearances.

We did not check bug IDs defined in Bugzilla bug tracking system available for each project, but we check the effectiveness of this approximation method by random sample inspection. We checked several logs manually and ascertained that the existence of the words "fix" or "bug" means some kind of bug correction. On the other hand we only found few Bugzilla IDs. To be precise we computed the number of bug fix and not the number of bug. Metric data and bug data collection process is summarized in Figure 2. The metrics data is computed based on the source files of one release and the bug data is computed based on the log data between two releases.

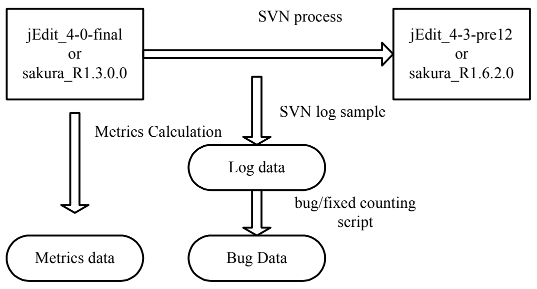
SVN process
jEdit_4-0-final or sakura_R1.3.0.0
jEdit_4-3-pre12 or sakura_R1.6.2.0
SVN log sample
Metrics Calculation
Log data
bug/fixed counting script
Metrics data
Bug Data

Figure 2: Extraction of the number of Bugs

We closely investigated the project data and found the following problems, and resolved these problems approximately as follows:

Table 1: Characteristics of the Used Projects

Project Name	The number of Developers	Used Language	Start Year	LOC	The number of Files
Sakura Editor	14	C++	2000	131,275	249
jEdit	144	Java	1999	169,107	538

- In C++, one file may include several classes and there are several header files. We considered only the files which include class definition. Bugs are counted per file, so if one file includes plural classes, then we assigned that bug count to each class equally.
- In Java, one file includes only one public class, but private classes and inner classes may be included. Bug count is computed per file and bug count per class is not computed, so we assigned that bug count to the public class.

We collected data based on the above approximation, and did classification experiments. At first we show the statistical data for bugs in Table 2 and 3. These data are computed based on the releases shown in Figure 2. As shown in Table 2, the number of bugs and the number of classes are different between Sakura Editor and jEdit. Table 3 shows the distribution of the bug count. The number of files which include more than one bugs are about thrice of the number of files which include no bug.

Table 3: The distribution of the bug count

project name	0	1-5	6-10	11-inf
Sakura Editor	33	38	4	5
jEdit	66	142	24	38

Table 4 shows the correlation coefficient between metrics used and the number of bugs (Metrics data are based on jEdit 4-0-final and Sakura R1.3.0.0 and bug data are based on jEdit (4-0-final to 4-3-pre12) and Sakura (R1.3.0.0 - R1.6.2.0). These values are similar to the correlation coefficient in PROMISE data base [2], so we consider that it is possible to predict error proneness by using these data.

Table 5 shows the average of each metric based on jEdit (4-3-pre12) and Sakura (R1.6.2.0). There are distinctive differences between these two metrics, but it is not obvious whether these differences are general or peculiar to these projects, and further analysis based on other projects are needed.

We have used the data mining tool Weka [6] to infer a prediction model, that is, binary classification scheme. We treated the existence of bug in each CLASS as TRUE, and the nonexistence as FALSE, and neglected the number of bugs, so we used the precision and recall defined based on TRUE value. The prediction is classified as the following four categories:

- It is true data and classified as true (TP)
- It is true data, but classified as false (FN)
- It is false data, but classified as true (FP)
- It is false data, and classified as false (TN)

Then, precision and recall will be computed as follows:

- The precision is the ratio of the number of files inferred as having positive bug count that has really positive bug count and the number of files inferred as having positive bug count
  $precision = TP/(TP + FP)$
- The recall is the ratio of the number of files inferred as having positive bug count that has really positive bug count and the number of files that really have positive bug count
  $recall = TP/(TP + FN)$

From the error prone prediction point of view, recall is more important than precision. A value close to one in recall means that almost all files which are error prone are identified. We evaluated the result by using recall, but showed the both values.

3. INTRA PROJECT BUG PREDICTION

At first we did intra project prediction experiments by using the same release. In Weka several algorithms can be used and we used landmark decision tree algorithm C4.5.

As shown in Figure 2, in order to use Weka for prediction, we need training data and test data, but we only collected metrics from one release, so we used stratified 10-fold cross validation method. We show the prediction result in Table 6(a).

Table 6: Intra Project Prediction 1

Project name	Precision	Recall
as-is data (a)		
Sakura Editor	0.733	0.702
jEdit	0.828	0.897
balanced data (b)		
Sakura Editor	0.659	0.630
jEdit	0.981	0.765

Our objective is to infer the error prone modules (files), and the existence of bug is represented as TRUE, so we used the precision and recall values based on TRUE value. Table 6(a) shows the result of experiments by using the as-is data. It shows good prediction, especially in jEdit, and shows the possibility of intra project error proneness prediction. As shown in Table 3, the number of TRUE, that is, the number of files which include more than one bug, is not equal with the number of FALSE, that is, the number of files which include no bug. It is about thrice. In general this unbalancing strengthens the effect of data whose count is larger, so we modified data to be balanced and tried the same experiment. We show the result in Table 6(b). In both projects, precision is improved, but recall is decreased. It is because precision depends mainly on FALSE value, on the other hand recall

Table 2: Statistical Data for Bugs

Project Name	the number of bugs	Average	Standard derivation	Maximum	Minimum	The number of classes
Sakura Editor	556	4.71	11.21	73	0	118
jEdit	2852	6.15	15.87	193	0	463

Table 4: Correlation Coefficient Between Metrics And the Number Of Bug

	WMC	DIT	NOC	CBO	RFC	LCOM	NPM	LOC
Sakura Editor	0.223	-0.259	-0.025	0.252	0.174	0.303	0.346	0.234
jEdit	0.120	0.037	-0.011	0.268	0.107	0.210	0.150	0.157

depends mainly on TRUE value. In this prediction, recall is more important than precision, so balancing has no clear effects.

Next we did true the intra projects prediction as shown in Figure 3 and showed the result in Table 7. Training data and test data are based on the different releases of the same projects. The recall values are smaller than the case of Table 6, especially in jEdit. We have not yet analyzed the reason.

Table 7: Intra Project Prediction 2

Project name	Precision	Recall
Sakura Editor	0.375	0.818
jEdit	0.575	0.598

4. INTER PROJECT BUG PREDICTION

We found that our data can predict the error prone files within own projects, but recall is not so good. We next do experiments in order to evaluate the possibility of inter project (in our case inter language) prediction using the same data set.

At first we evaluated whether there is significant difference between these two metric data. As shown in Table 4 and 5, these values have differences. We validated the difference by using classification techniques. We try to classify these two datasets (the dataset for sakura consists of metrics data and the literal "sakura", and the dataset for jedit consists of the metrics data and the literal "jEdit") by using the C4.5 method and got over 90% precision values, that is, over 90% of dataset are classified correctly as "sakura" or "jEdit", so we conclude that there is significant difference between these two data and the hypothesis 1 is validated.

Table 8: Inter project prediction

Training project	Test Project	Precision	Recall
as-is data (a)			
Sakura Editor	jEdit	0.872	0.402
jEdit	Sakura Editor	0.622	0.596
Compensated data (b)			
Sakura Editor	jEdit	0.842	0.549
jEdit	Sakura Editor	0.625	0.745

Next we did inter language prediction experiments, and show the result in Table8(a). In this case, the training data is collected from Sakura Editor (jEdit) and the test data is collected from jEdit (Sakura Editor). These results show the possibility of inter language reuse, especially in the case of jEdit to Sakura Editor (the recall value is 0.817). In this case the recall value is larger than the case of within project reuse.

Next we consider the adaptation of this model. There are two possibilities of this adaptation:

- Metrics compensation: To adjust the test data.
- Model schema adaptation: In C4.5, generated schema is decision tree, so each judgment factor may be adapted.

In this paper, we only tried the first approach. As shown in Table 5, the averages of each metric value are different from jEdit to Sakura Editor, so we thought that these values need to be normalized. We considered the following compensation method: we suppose that we try to predict error prone files of project A by using the training data of project B, then the test data will be compensated as follows:

(each compensated metric value of project A) = (each metric value of project A) * (average value of that metric of project B) / (average value of that metric of project A)

The compensated values become similar to the learning data, so the availability must be increased. We show the prediction result based on these compensated data in Table 8(b). Both precision and recall are improved, especially, recall is improved so much (in the case of jEdit to Sakura, the recall value is 0.983), so this compensation is effective. As a result, the hypothesis 2 is validated.

5. RELATED RESEARCHES

There are several hundred papers about effort prediction [12]. In effort prediction, language based factor is proposed and used for the adaptation of the prediction model, for example in COCOMO [9]. Metrics are used not only for effort prediction but for quality prediction, for example, error prone module prediction [7, 22, 11, 21, 17, 18, 23, 10]. Basili, etl [7] showed the possibility of error prone module prediction by using C&K metrics. Their target language is C++ and they pointed out the necessity of researches for other languages. On the other hand Subramanyam [21] evaluated the error prone prediction for C++ and Java project. They indicated the difference of each metric. Menzies [17] also evaluated the error proneness for C and Java projects.

Table 5: Average of Each Metric

	WMC	DIT	NOC	CBO	RFC	LCOM	NPM	LOC
Sakura Editor	17.70	0.36	0.39	5.06	26.84	57.22	10.14	576.21
jEdit	10.94	2.27	0.71	13.40	155.00	46.18	7.33	196.53

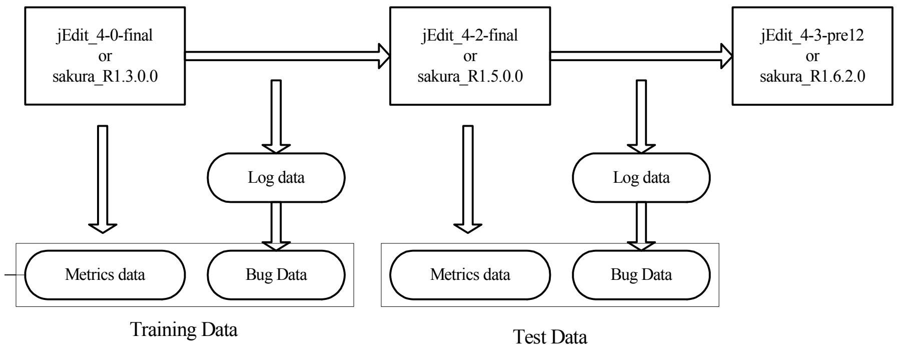
jEdit_4-0-final or sakura_R1.3.0.0
jEdit_4-2-final or sakura_R1.5.0.0
jEdit_4-3-pre12 or sakura_R1.6.2.0
Log data
Log data
Metrics data
Bug Data
Metrics data
Bug Data
Training Data
Test Data

Figure 3: Intra Projects Error Prediction 2

These researches have shown the possibility of error prone prediction for each language and the necessity of different treatment for each language, but do not evaluate the possibility of prediction model reuse. In this paper we have shown the possibility of this reuse. Nagappan [18] evaluated the possibility of inter project reuse of error prone prediction and negated the possibility.

On the other hand, there are several researches about inter project reuse or cross-company effort prediction [14, 13, 16, 15] and defect prediction [18, 19]. MacDonell [14] showed the possibility of inter company reuse of an effort prediction model, and Zimmermann [23] showed the possibility of inter version reuse of defects prediction for Eclipse.

6. CONCLUSIONS

We did inter language prediction experiments for error proneness and have gotten the following results:

- In the case of similar domain and similar size, it is possible to reuse the prediction model between languages, but precision/recall is not so high.
- Compensation based on metric value is effective, and it improves the recall value.
- Reuse direction has much effect on recall values.

Past researches considered only how to improve the precision/recall for intra project prediction [10]. In order to construct prediction model, training data is needed, and that training data consists of the data of independent variables and that of dependent variables. In error prone prediction, independent variables represent source code metrics and dependent variable represents bug count, but the collection of bug count is not so easy and it needs some time span for bug collection. If reuse becomes possible, then needed data are only metric data and the availability of these prediction models will increase. In this paper we show the possibility of such reuse. Our result based on only two projects, so the generality is not clear. Our final goal is to construct general error proneness prediction model shown in Figure 4, which includes domain/language dependent compensation methods and factors. We need to do further researches considering the following problems in order to realize this model.

- The reason of impact of reuse direction on recall values (from A to B is good, but B to A is bad) needs to be cleared and we consider some resolving method.
- In order to generalize our method, we will evaluate by using other projects whose domain is editor, and evaluate the reusability of prediction models.
- We only considered metric compensation (compensation 1 in Figure 4), but prediction schema may be compensated (compensation 2 in Figure 4). We used decision tree scheme, so we can adapt the decision tree parameter. We will consider schema adaptation.

7. REFERENCES

[1] jedit. `http://sourceforge.net/projects/jedit/`.
[2] promise. `http://promisedata.org/repository/`.
[3] sakura-editor. `http://sourceforge.net/projects/sakura-editor/`.
[4] scitools. `http://www.scitools.com/index.php`.
[5] sourceforge. `http://sourceforge.net/`.
[6] weka. `http://www.cs.waikato.ac.nz/~ml/weka/index.html`.
[7] V. Basili, L. Briand, and W. Melo. A validation of object oriented design metrics as quality indicators. *IEEE Trans. Software Eng.*, 22(10):751–761, 1996.
[8] R. M. Bell, T. J. Ostrand, and E. J. Weyuker. Looking for bugs in all the right places. In *ISSTA*, pages 61–71. ACM, 2006.
[9] B. W. Boehm. *Software Cost Estimation With Cocomo II.* Prentice-Hall, Reading, 2000.
[10] G. Denaro and M. Pezze. An emprirical evaluation of fault-proness models. In *ICSE*. ACM, 2002.

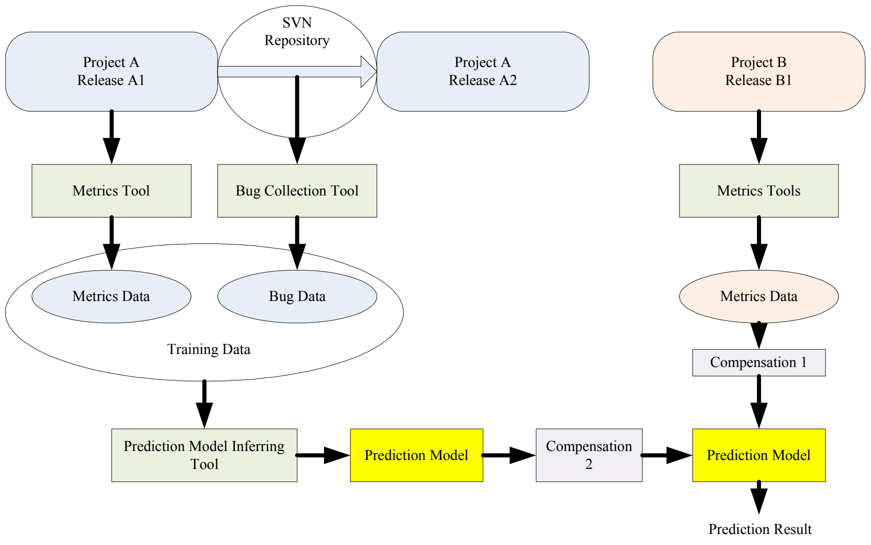

Figure 4: General Error Prone Module Prediction Schema

[11] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open software for fault prediction. *IEEE Trans. on Soft. Eng.*, 31(10):897–910, 2005.

[12] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 33(1):33–53, 2007.

[13] B. A. Kitchenham, E. Mendes, and G. H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 33(5):316–329, 2007.

[14] S. G. MacDonell and M. J. Shepperd. Comparing local and global software effort estimation models . reflections on a systematic review. In *International Symposium on Empirical Software Engineering and Measurement*, pages 401–409. IEEE, 2007.

[15] E. Mendes and B. Kitchenham. Further comparison of cross-company and within-company effort estimation models for web applications. In *Metrics*. IEEE, 2004.

[16] E. Mendes, C. Lokan, R. Harrison, and C. Triggs. A replicated comparison of cross-company and within-company effort estimation models using the isbsg database. In *Metrics*. IEEE, 2005.

[17] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Trans. on Soft. Eng.*, 33(1):2–13, 2007.

[18] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *ICSE*, pages 452–461, 2006.

[19] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 31(4):340–355, 2005.

[20] S.R.Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. on Soft. Eng.*, 20(6):476–493, 1994.

[21] R. Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. on Soft. Eng.*, 29(4):297–310, 2003.

[22] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Adapting a fault prediction models to allow widespread usage. In *promise*, 2006.

[23] T. Zimmermann, R. Premaraj, and A. Zellar. Predicting defects for eclipse. In *promise*, 2007.