International Journal of Computer Applications (0975 – 8887)
Volume 74– No.8, July 2013

Cross Company and within Company Fault Prediction using Object Oriented Metrics

Pradeep Singh
National Institute of Technology

Shrish Verma
National Institute of Technology

O P Vyas
Indian Institute of information Technology

ABSTRACT

This paper investigates fault predictions in the cross-project context focusing on the object oriented metrics for the organizations that do not track fault related data. In this study, empirical analysis is carried out to validate object-oriented Chidamber and Kemerer (CK) design metrics for cross project fault prediction. The machine learning techniques used for evaluation are J48, NB, SVM, RF, K-NN and DT. The results indicate CK metrics can be used as initial guideline for the projects where no previous fault data is available. Overall, the results of cross company is comparable to the within company data learning. Our analysis is in favour of reusability in object oriented technology and it has been empirically shown that object oriented metric data can be used for cross company fault prediction in initial stage when previous fault data of the project is not available.

Keywords

Fault prediction, cross company, Software metric, open source software.

1. INTRODUCTION

Modern software engineering and programming practices stress to develop similar products using reusable core assets. Modularization or component-based software designs, architecture and implementation are the basic technique for building software with new or reusable parts. The general aim of reusability is to enhance quality and to minimize the development effort and time. Software reuse can result in substantial savings in the development costs as well as in development of low complexity end-products that are relatively small in size. In order to increase productivity and quality, organizations develops a module once, verifies that it functions correctly and properly, and then reuses it in different applications where the same functionality is required. In the age of open source development it is quite possible that organizations can also make use of reusable components from outside their organization than within the organization. The software reuse concept is probably the most significant part of object-oriented based information system development. It is hence reasonable to use such cross company data which are developed using object oriented methodology. According to Kitchenham et.al. [1] : “The time required to collect enough data on past projects from within a company may be prohibitive. Collecting within-company data may take so long that technologies change and older projects do not represent current practice.” Object-oriented development methodology is greatly used in software industry and many design metrics of object-oriented programs have been proposed for fault prediction, but there is no cross company investigation has been reported so far. In this study, empirical analysis is carried out to validate object-oriented design metrics for cross project fault prediction .The Chidamber and Kemerer metrics suite is adopted to predict the faults in the projects using same and cross company data. We use CK metric suite from software developed by different organization, using different object oriented language. The machine learning techniques used for evaluation are statistical, J48, NB, SVM, RF, K-NN and DT. The result indicates that CK metrics can be used as initial guideline for the projects where no fault data is available. Overall, the results of cross company is comparable to the within company data learning.

Software fault prediction using various techniques on software repository for predicting the fault-prone software modules is of a great interest among the software testing researchers and industry professionals for reducing the cost occurring in software testing. Researchers have used metric based classification for software components as fault-prone and non-fault-prone [6][7]. Researchers and engineers have used static design metrics of the programs for this purpose. Many researchers have explored issues like the relative merits of McCabes cyclomatic complexity, Halsteads software science measures, and lines of code counts for building fault predictors [6,7,16]. After object-oriented programming dominated software development, a vast variety of design metrics have been adapted for estimating the quality of object-oriented programs. Chidamber and Kemerer[4] introduced their OO design and complexity metrics and demonstrated the strong impact on software quality. The CK metrics suite invoked great enthusiasm among researchers and software engineers, and a great amount of empirical studies have been conducted to evaluate those metrics. In this study, data from the industry is used to analyze the relationships between CK metrics and faults in the OO programs.

Metrics data can be computed by using automatic tools, but it is not so easy to collect bug data. In the present work, we try to reuse the fault data of one project to generated prediction model for another project.

To achieve this, the metrics and bug data computed from C++ and Java projects, then selection of CK metric for both project are used to create fault prediction model. Fault prediction models focuses on predicting the fault-prone modules precisely and helps software manager and testers to allocate limited resources in testing and maintenance Studies on this issue, usually trained predictors from data of historical releases in the same project (i.e., faults distributional data and software metrics such as static code features, code change histories, and process metrics) and predicted faults in the upcoming releases, or reported the results of cross-validation on the same data set.

To build a fault predictor we need to extract the fault and software code data from the software repositories of the same project that is, training data for the predictor. However, sometimes in real practice, such faulty chronological data is not always accessible, because either it does not yet exist due to the starting of project or was not well collected. That means, having a fault prediction model for those companies which do not track fault related to the same project is impossible initially in some cases. On the other hand, open source repositories provides plenty of public fault related data. A potential way of predicting faults for projects of these companies without historical fault data is to make use of these public and open source projects as training data. Cross-project fault prediction refers to predicting faults in a project using prediction models trained from historical data of other projects [9,10]. There are some studies focusing on this issue and their results show that cross-project fault prediction is still a serious challenge [9,11].For machine learning based predictions, the effect of a predictor depends on two factors: the training data and the learning algorithm. Consequently there are two potential ways for cross-project fault prediction: first one is collecting the best suitable training data for the project to be predicted, ideally we may find a training data presenting the same fault pattern with the target project, which will lead to acceptable prediction results and the other is by using learning algorithms with high ability to classify the fault prone and non-fault prone class .This method assumes that there exists a general fault pattern between different data sets. If we can learn this pattern successfully, we can predict faults in one projects based on the data of other project. In this paper, we investigate the empirical evidences to answer the following questions in context of cross-project fault prediction:

Can raining data from different projects developed in different environment by using different object oriented language provide acceptable prediction results by using CK metric?

Does training data from the same project always lead to better prediction results than training data from other projects?

The rest of this paper is structured as follows: Sect. 2 summarizes some related work; Sect. 3 describes the data set, and the performance evaluation criteria; Sect. 4 describes the learning algorithms and the experiments we conducted. Sect. 5 explains threats to validity .Finally we conclude this study in Sect. 6.

2. RELATED WORK

Software testing is one of the most important and critical quality assurance activities. It is a time consuming and labor-intensive activity in software development life cycle while resources allocated for testing are usually limited[11,12]. Fault prediction has been proved to be effective for optimizing testing resource allocation by identifying the modules that are more likely to be fault prone prior to testing [14]. In the past decade, various fault prediction models have been proposed and machine learning techniques have become more and more popular in constructing fault prediction models [6, 15,16]. Menzies [6] also evaluated the error proneness for C and Java projects. However, most prediction models reported are intra-project applicable, i.e., learning from data of historical releases and then applying to the upcoming release in the same project. The application field of these models is restricted for data of historical releases is unavailable sometimes. This research aims to extend the application field of fault prediction. It is unlike most previous software fault prediction research for its focus on a cross-project context. The problem of predicting faults in a cross-project context drew the attention of many researchers in recent years. To the best of our knowledge, studies on cross project fault prediction do not show a conclusive picture so far. However, there are considerable studies focusing on this issue. Zimmermann et al. [17] used 12 real world application for cross-project fault predictions and found that only few predictions worked successfully .That means cross-project fault prediction will fail in most cases if not selecting training data carefully. They also found that cross-project fault prediction is not symmetry. For example, data of Firefox can predict Internet Explorer faults well (Precision equal to 76.47% and Recall equal to 81.25%) but the opposite direction does not work (Recall equal to 4.12%). They argued that characteristics of data and process are vital factors for the effective fault prediction of cross-project rather than domain. Consequently they concluded that simply using historical data of projects in the same domain does not lead to good prediction results. Zimmermann et al. pointed out that cross-project fault prediction is a serious challenge and more attention should be paid to this problem. Turhan et al. [11] also studied a cross-company fault prediction problem , that is, using data from other companies to build fault predictors for local projects. In their experiments they used only static code features to build fault predictors. They concluded that cross-company data increase the probability of fault detection (pd) at the cost of increasing false positive rate (pf). Their experimental results show that nearest neighbor filtering can help to reduce pf when using cross-company data.

3. DATA SET USED

Our project data are taken from software developed on different standards, language and location .KC1 developed in North America (NASA) and JEdit is an open source software. Therefore, the code features that are available for each project vary. KC1 have 95 features available, we extracted only the CK metric from both different projects. Table 1 shows the common features for both sources. The data set KC1 from the NASA IV and V Facility Metrics Data Program data repository (http://mdp.ivv.nasa.org), which is comprised of 43 KSLOC of C++ code for a ground system. There are 145 classes and altogether 2107 methods. The faults information in the original data set is at method level, while metrics information is at class level. For this study, those files have to be combined to get all class level information. We generate all class level information and validate it with the above data resource. Descriptive information about this public data set is listed in Table 1. Since six CK metrics are examined to evaluate their impact on the quality of the code, only related information is included in the Table 1. JEdit a text editor developed using Java language. It is an open source project and its software and the source code is freely available. The LOC of JEdit is 169,107. The number of developers involved in this project was 144. The project was started in 1999. The number of bugs was computed using SVN repositories. The release point for the project was identified in 2002. The log data from that point to 2007 was collected. The word bug or fixed was counted. Details of bug collection process can be found in [19]. The intention of this study is to validate and compare the influence of CK metrics on the cross company software fault predictability.

CK metrics: In CK metrics suite [4], six design and complexity metrics are used to represent the characteristics of the code:

WMC (Weighted Methods per Class): The sum of normalized complexity of all methods in a given class. Usually the method complexity is measured using cyclomatic complexity. Consider a class X1 with methods M_1 ,M_2,…,M_n that are given in the class. Let C_1,C_2,…,C_n be the complexities of the methods. The WMC is calculated as

WMC= $\sum_{i=1}^{n} C_i$

DIT (Depth of Inheritance Tree): The maximum length from the root to a given class in the inheritance hierarchy

NOC (Number of Children): The number of immediate subclasses of a given class in a hierarchy.

RFC (Response For a Class): The number of methods implemented in a given class that can be invoked by a received message

CBO (Coupling Between Object Classes): CBO for a class is the count of the number of other class to which it is coupled. It is the number of classes that use the member functions and/or the instance variables of a given class .The more independent a class is, the easier it is to reuse in different applications.

LCOM (Lack of Cohesion on Methods): for each instance variable calculate the percentage of methods using it, then the average percentage for all variables subtracted from 100%.

These CK metrics express the quantifiable and measurable characteristics of an OO program, such as complexity, cohesion and coupling. For general design environment, technically these CK metrics should be kept at a reasonable level.

Table I: Descriptive information of metrics Metric of KC1 and JEdit

	Max		Mean		St dev	
	KC1	JEdit	Mean	JEdit	KC1	JEdit
CBO	24	105	8.3	12.64	6.37	14.13
DIT	7	7	2.0	2.49	1.258	1.97
LCOM	100	100	68.7	46.23	36.889	33.51
NOC	5	35	0.214	0.715	0.699	3.1
RFC	222	843	34.4	174.97	36.203	269.5
WMC	100	407	17.4	11.72	17.449	31.20

Accuracy measures

The accuracy and performance of prediction models for two-class problem, faulty or not faulty is typically evaluated using a confusion matrix. A confusion matrix contains information about actual and predicted classifications done by a classification system. In this study, we used the commonly used prediction performance measures: probability of detection (pd), probability of false alarm (pf), precision (prec), recall and f-measure to evaluate and compare prediction models quantitatively. These measures are derived from the confusion matrix.

A confusion matrix

Predicted	Actual	Faulty	Not faulty
Faulty		TP	FP
Not faulty		FN	TN

False alarms, pf, should be 0, meaning that the predictor should never label a fault-free module as faulty. In general, an increase in pd would also increase pf rates since the model triggers more often to achieve the ideal case. Precision is also known as correctness. It is defined as the ratio of the number of modules correctly predicted as faulty to the total number of modules predicted as faulty.

$$\text{Recall}= pd = TP/ (TP+FN)$$

$$pf = FP/(FP+TN)$$

$$\text{Precision}=TP/ (TP+FP)$$

The precision is the ratio of the number of files inferred as having positive bug count that has really positive bug count. The higher the precision, the less effort is required for testing and inspection. It has a strong relation with pd and pf, such that when pd is fixed for a dataset, pf rate is controlled by precision and the class distribution of the data [6].

F-measure considers both precision and recall equally important by taking their harmonic mean. It is calculated as follows:

$$\text{F - measure} = \frac{(\beta^2 + 1) \times \Pr ecision * \operatorname{Re} call}{\beta^2 \times \Pr ecision + \operatorname{Re} call}$$

The F-measure has been widely used as a measure in the field of Information Retrieval and Data Mining. It integrates Recall and Precision in a single indicator. Parameter β indicates the weight assigned to Recall and it assumes any non-negative value. Higher β value means higher weight. If β is equal to 1, Recall and Precision are given the equal weight. Here in our experiments, we treat Recall and Precision equally, so we set β to 1.The higher the quality of the predictor, the higher the F-measure.

4. EXPERIMENT DESIGN

We employ six machine learning algorithms to construct predictors. They are J48 (a C4.5 decision tree), Naïve Bayes (NB), Support Vector Machine (SVM), Random forest (RF), K-NN and Decision Table (DT). These 6 learners have been widely used in the context of fault prediction [6][7][19].

The J48 is the java version of C4.5 decision tree learner. C4.5 is an extension of the ID3 algorithm. It builds the tree structure from the training data by using the concept of information entropy. Leaves in the tree structure represent classifications and branches represent judgment rules. More details about the C4.5 Decision Tree learner can be found in Quinlan [18].The Naive Bayes learner is based on probability theory and assumes that features of the data set are independent of each other. Although the independence assumption is often violated in reality, the Naive Bayes learner has been proved to be effective for fault prediction [6]. Support vector machines (SVM) are kernel based learning algorithm introduced by Vapnik[20] using the Structural Risk Minimization (SRM) principle which minimizes the generalization error, i.e., true error on unseen examples. The basic SVM classifier deals with two-class pattern recognition problems, in which the data are separated by the optimal hyperplane defined by a number of support vectors. The regularization parameter (C) was set at 1; the kernel function used was Gaussian (RBF); Comparative studies conducted by Lessmann et al. [7] shows that the SVM learner performs equally with the Naïve Bayes learner in the context of fault prediction. Random Forest ensemble introduced by Breiman [22] uses a large number of individual, unpruned decision trees which are created by randomizing the split at each node of the decision tree. Each tree is likely to be less accurate than a tree created with the exact splits. But, by combining several of these "approximate" trees in an ensemble, can improve the accuracy and often it is doing better than a single tree with exact splits. In k-nearest neighbor, a test sample is compared with existing ones by using a distance metric and the majority class of the closest k neighbors is assigned to the test case.

The distance between two samples can be computed by using any distance metric like Euclidean and Manhattan, we have used Euclidean distance. The Decision Table learner is a rule based learner whose result can be easily understood. More details of the comparison of fault prediction models based on different learners can be found in Lessmann et al. [7]. According to the suggestion of [6], we do not introduce any attribute selection techniques when constructing fault prediction models in our experiments. Previous studies show that cross-project fault prediction remains a challenging issue. We need to verify that whether prediction results provided by cross-project data are acceptable, in order to validate the above hypotheses we need at least two projects whose target domain are different and the languages is also different but both are of object oriented programming. Fig 1 shows the complete process of cross company fault prediction model using CK metric.

The pseudo code given in fig2, for within company (WC) and cross company (CC) analysis for the projects KC1 and JEdit. Data came from one systems written in "C++" and the other systems was written in JAVA. For cross-company data, an industrial practitioner may not have access to detailed meta-knowledge (e.g. whether it was developed in "C++" or JAVA). They may only be aware that data, from an unknown source, are available for download. The projects data come from different sources and, hence, have different features. For this analysis, we have used only the six CK metric which are common in both the projects.

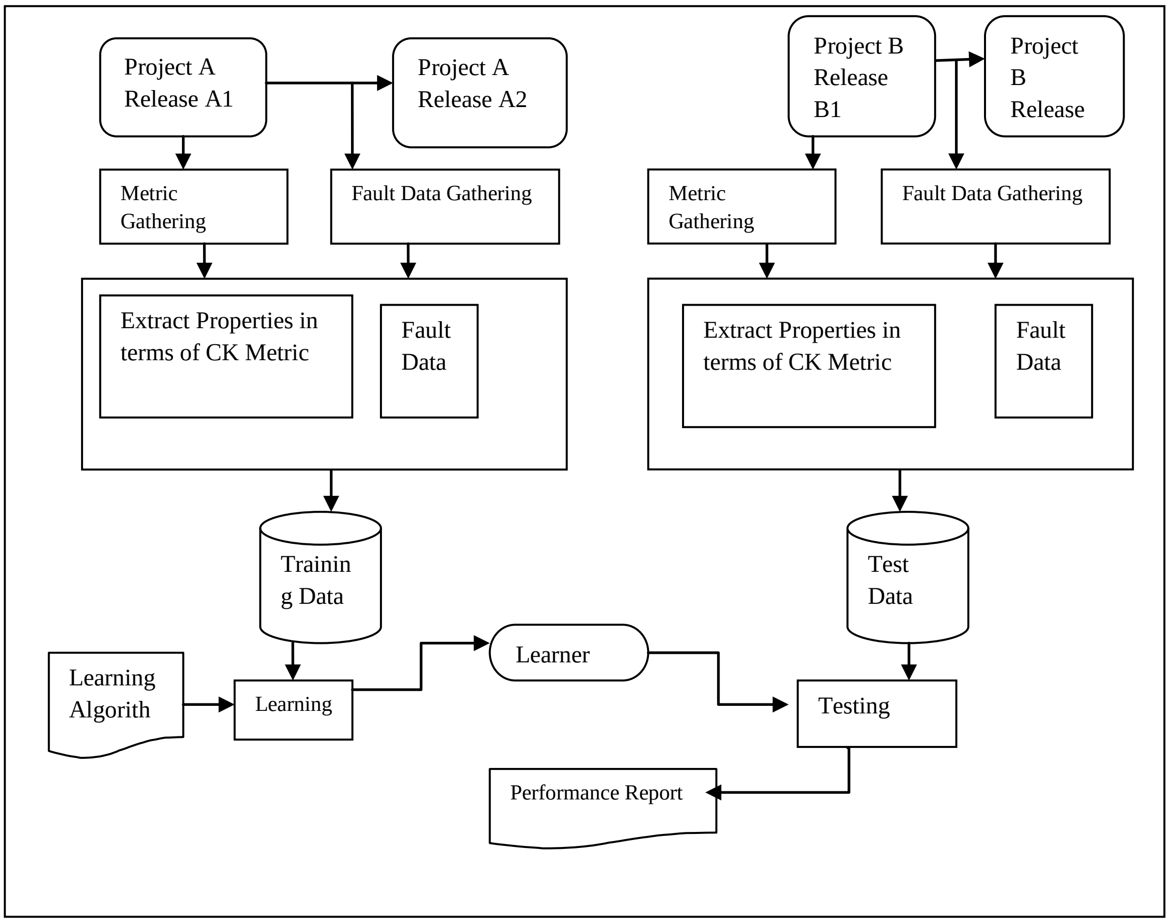

Fig 1 Cross project fault prediction model using CK metrics

```
DATA = [KC1, JEdit] // Different data from different domain and
developed using different language all available data
LEARNER = [J48,Naive Bayes, SVM, RF,KNN,DT]
//   fault predictor
C_FEATURES:-Find CK features IN DATA
FOR EACH data IN DATA
data = Select C_FEATURES in data // use common features
END
FOR EACH data in DATA
CC_TRAIN = DATA - data // cross company training data
WC_TRAIN = random 90% of data // within company training data
TEST = data - WC_TRAIN // shared test data
//construct predictor from CC data
CC_PREDICTOR = Train LEARNER with CC_TRAIN
// construct predictor from WC data
WC_PREDICTOR = Train LEARNER with WC_TRAIN
//Evaluate both predictors on the same test data
[cc_pd, cc_pf, cc_prec,cc_fmes] = CC_PREDICTOR on TEST
[wc_pd, wc_pf, wc_prec,wc_fmes] = WC_PREDICTOR on     TEST
END
END
```

Fig 2 Pseudo code for Cross Company Analysis

Firstly we performed the prediction experiment by using training data within the company by using M*N-way cross validation where both M and N are selected as 10 [44]. We create 10 stratified bins: 9 of these 10 bins are used as training sets and the last one is used as the test set. We randomize the dataset M = 10 times and create N = 10 sets in each iteration. Next we did cross company and inter language prediction experiments. The table 2 to table 7 shows the result of within company and cross company prediction. In case of cross company there are two scenarios first we collected the training data of KC1 and extracted CK metrics and related bug data and the test data set is JEdit with the Ck metric. In second scenario we have used JEdit CK metric data set for training and KC1for testing. These projects are developed in different object oriented language and are of different domain. For J48 we construct a decision tree on the train-test-result instances generated. If the decision tree can identify those successful cross-project fault predictions precisely, it means that distributional characteristics of data set are related to prediction results in the cross-project context.

Table II: Prediction performance measures using pd

Cross company							
Training Project	Test Project	j48	NB	SVM	RF	K-NN	DT
KC1	KC1	0.633	0.583	0	0.583	0.55	0.633
JEdit	JEdit	0.746	0.53	0.015	0.672	0.619	0.642
KC1	JEdit	0.47	0.575	0	0.328	0.552	0.343
JEdit	KC1	0.7	0.367	0	0.667	0.467	0.583

Considering cross-project predictions when JEdit is used for training and test set of KC1the prediction are successful (see Table 2), we think the performance of this decision tree is fairly high. The high performance of the decision tree indicates that the distributional characteristics of training set and test set are related to results of cross-project fault predictions. In case of RF the probability of fault detection is even better than the result when training is done with within the project data set. When learner is J48, the pd is 70% in case of cross project and 63.3% for within company for KC1 test data. Form the result it is evident that open source project JEdit when used for training and KC1 for testing which was developed in a process oriented approach by a cost driven government entity, the cross project fault prediction is better for J48 and RF.

Table III Prediction performance measures using   pf

Cross company							
Training Project	Test Project	j48	NB	SVM	RF	K-NN	DT
KC1	KC1	0.353	0.259	0	0.247	0.282	0.376
JEdit	JEdit	0.25	0.15	0	0.286	0.271	0.307
KC1	JEdit	0.157	0.193	0	0.057	0.307	0.05
JEdit	KC1	0.271	0.129	0	0.353	0.224	0.529

This paper provides empirical evidences and interesting results for both software quality assurance and computational intelligence communities. Learner J48, NB and K-NN produced the less pf when JEdit is used for training and KC1 is used for test. In case of J48, DT and RF, when training is done by KC1 and testing is done on open source data JEdit, the pf is very less compared to within project. However, in case of SVM, there are no changes in the performance in pf.

Table IV: Prediction performance measures using Precision

Cross company							
Training Project	Test Project	j48	NB	SVM	RF	K-NN	DT
KC1	KC1	0.559	0.614	0	0.625	0.579	0.543
JEdit	JEdit	0.741	0.772	1	0.692	0.686	0.667
KC1	JEdit	0.741	0.74	0	0.846	0.632	0.868
JEdit	KC1	0.646	0.667	0	0.571	0.596	0.438

Average predictors based on learning methods NB and J48 can provide the best prediction results with (Precision) equal to 0.7035 and 0.639 respectively in case of cross company whereas the average within company performance in terms of precision is 0.693 and 0.65 respectively. From the results it is evident that the results using the CK metrics for fault prediction in case of cross company is better than the within company   in case of NB learner and is competitive in case of J48.

Table V Prediction performance measures using F-measure

Cross company							
Training Project	Test Project	j48	NB	SVM	RF	K-NN	DT
KC1	KC1	0.594	0.598	0	0.603	0.564	0.585
JEdit	JEdit	0.743	0.628	0.029	0.682	0.651	0.654
KC1	JEdit	0.575	0.647	0	0.473	0.59	0.492
JEdit	KC1	0.672	0.473	0	0.615	0.523	0.5

From Table 5, it can be observed that for F-measure, J48, NB, and RF achieves higher F-measure than within company data . NB achieved f-measure (64.7%), in case of training by KC1 and test set is JEdit but this is not higher than the performance of learner J48, when training by JEdit and test set is KC1.

5. THREATS TO VALIDITY

Due to the many factors that affect software development: the type of application domain; the programming capability of the individual programmers involved in system development; development practices; the variation in measurement practices; and the quality of the measurements and instruments used to collect the data, and consequently, software quality, controlled experiments for evaluating the usefulness of empirical models are not practical. We adopted a case study approach in the empirical investigations and presented in this paper. Software engineering community demands that the subject of an empirical software study have the following characteristics [23]:
. Developed by a group, and not by an individual.
. Be as large as industry-size projects, and not a toy problem.
. Developed by professionals, and not by students.
. Developed in an industry/government organization setting, and not in a laboratory.
We note that our case studies fulfill all of the above criteria. The software systems investigated in our study were developed by software professionals and government software development organizations. In addition, each software was developed to address a real-world problem. Another point is that descriptions of software modules only in terms CK metric can overlook some important aspects of software including: the type of application domain and other important static code attributes. Several threats to the external validity of our study may restrict the generalization of our results. Some of the limitations are: The degree to which the results of our study can be generalized to other research settings is questionable. The reason is that the system severity of faults is not taken into account for this study. There may be different types and different number of faults which can leave the system in various states e.g. a failure that is caused by a fault may lead to a system crash or a failure to process a file. The project studied lie between 43 KSLOC and 169 KSLOC. These programs are small as compared to large industry systems. The prediction capabilities of the studied CK object oriented design metrics may results very different in larger programs. The conceptual complexity of these systems was rather limited. Again, many different problems may arise in more complex systems. Though these results provide guidance for future research on the use of machine learning methods to find the impact of OO metrics on fault proneness for cross company, further validations are necessary with different systems to draw further stronger conclusions.

6. CONCLUSION

Cross-project fault prediction is important for projects without previous fault data. It extends the application field of fault prediction models, e.g., predicting faults in the first release of a new project. However, empirical evidences show that cross-project fault predictions only work in a few cases. In this study, we reported results from experiments of cross-project fault prediction with regard to object oriented CK metrics. The experiments were conducted on two data sets collected from two different source projects developed in two different object oriented programming language for fault prediction . NASA project which we have used for cross-company follow stringent ISO-9001 industrial practices imposed by NASA and the other project is open source software .Our study reveals that when training is done by the NASA project for open source software then the false alarm rate (pf) decreases and for probability of detecting (pd) increases when training is done by open source software for NASA project. The result shows that in the case of different domain and different size, it is possible to reuse the prediction model between languages and projects of different companies. Finally it is evident from the result that CK based object oriented metric value is effective for cross project fault prediction. In overall performance evaluation J48 has performed better that the other classifiers in pd, recall and F-measure. It is also evident that the results in terms of precision using CK metrics for fault prediction of cross company is better than the within company in case of NB learner and is competitive in case of J48.Our analysis of cross company data and the proposed methodology allows the construction of fault predictors even for companies where no local fault data is available. Our final goal is to construct general fault prediction model for successful cross project fault prediction.

7. REFERENCES

[1] Basili, V.R. and B.T. Perricone, 1984. Software errors and complexity: An empirical investigation. Commun. ACM, 27: 42-52. http://portal.acm.org/citation.cfm?id=2085.

[2] Halstead, M.H., 1977. Elements of Software Science. 1st Edn., Elsevier North Holland, New York, ISBN: 10: 0444002057 pp: 127.

[3] McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng., 2: 308-320. DOI: 10.1109/TSE.1976.233837.

[4] Chidamber, S.R. and C.F. Kemerer, 1994. A metrics suite for object-oriented design. IEEE Trans. Software Eng., 20: 476-493. DOI:10.1109/32.295895.

[5] Basili, V.R., L.C. Briand and W.L. Melo, 1996. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Software Eng., 22: 751-761. DOI: 10.1109/32.544352

[6] Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn fault predictors. IEEE Trans. Softw. Eng. 33(1), 2–13 (2007b)

[7] Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software fault prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. 34(4), 485–496 (2008)

[8] D’Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, pp. 31–41 (2010)

[9] Zimmermann, T., Nagappan, N., Gall, H.: Cross-project fault prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 91–100 (2009)

[10] Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the InternationalWorkshop on Predictive Models in Software Engineering, pp. 19–24 (2008)

[11] Turhan, B., Menzies, T., Bener, A.: On the relative value of cross-company and within_company data for fault prediction. Empir. Softw. Eng. 14(5), 540–578 (2009)

[12] Ostrand, T.J., Weyuker, E.J., Bell, R.M.: Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. 31(4), 340–355 (2005)

[13] Boetticher, G., Menzies, T., Ostrand, T.J.: PROMISE repository of empirical software engineering data. http://promisedata.org/repository (2007). Accessed 12 December 2010

[14] Tosun, A., Bener, A., Kale, R.: AI-based software fault predictors: applications and benefits in a case study. In: Proceedings of the 22th Innovative Applications of Artificial Intelligence Conference, pp. 1748–1755 (2010)

[15] Nagappan, N., Ball, T.: Use of relative code churn measures to predict system fault density. In: Proceedings of the 27th International Conference on Software Engineering, pp. 284–292 (2005)

[16] Catal, C., Diri, B.: A systematic review of software fault prediction studies. Expert Syst. Appl. 36(4), 7346–7354 (2009)

[17] Zimmermann, T., Nagappan, N., Gall, H.: Cross-project fault prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 91–100 (2009)

[18] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo (1993)

[19] Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the InternationalWorkshop on Predictive Models in Software Engineering, pp. 19–24 (2008)

[20] Vapnik, V., 1995. The Nature of Statistical Learning Theory. Springer, New York.

[21] D. Aha, D. Kibler (1991). Instance-based learning algorithms. Machine Learning. 6:37-66.

[22] Breiman, L., 2001. Random forests. Machine Learning 45, 5–32.

[23] C C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A.Wesslen, Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, 2000.