International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:3, 2015

Cross Project Software Fault Prediction at Design Phase

Pradeep Singh, Shrish Verma

***Abstract*—**Software fault prediction models are created by using the source code, processed metrics from the same or previous version of code and related fault data. Some company do not store and keep track of all artifacts which are required for software fault prediction. To construct fault prediction model for such company, the training data from the other projects can be one potential solution. Earlier we predicted the fault the less cost it requires to correct. The training data consists of metrics data and related fault data at function/module level. This paper investigates fault predictions at early stage using the cross-project data focusing on the design metrics. In this study, empirical analysis is carried out to validate design metrics for cross project fault prediction. The machine learning techniques used for evaluation is Naïve Bayes. The design phase metrics of other projects can be used as initial guideline for the projects where no previous fault data is available. We analyze seven datasets from NASA Metrics Data Program which offer design as well as code metrics. Overall, the results of cross project is comparable to the within company data learning.

***Keywords*—**Software Metrics, Fault prediction, Cross project, Within project.

I. Introduction

SOFTWARE fault prediction focuses on identifying the fault-prone modules precisely and helps in allocation of limited resources in software testing and maintenance. This paper considers the task of identifying fault prone software modules at design phase by means of design phase metric-using cross project fault data. It has been pragmatic that the majority of faults in software are contained in a small number of modules [3], [4]. Consequently, an early identification of these modules at design phase facilitates an efficient allocation of testing resources and may enable architectural improvements by suggesting a more rigorous design for mission critical systems. Empirical experiments on this issue usually trains predictors from the data of the previous releases of same project and predicts defects in the upcoming releases, or reported the results of cross-validation on the same data set [2], [5], [6]. To construct such a fault predictor one needs to collect, manage and process software repositories of the same project. However, in practice, such kind of historical data is not all the time available, because either it does not yet exist or was not well collected and managed [7], [8]. That means fault prediction based on historical data of the some project is impossible where no previous data of same project is available. On the other hand, there are many public defect data sets available, especially in the open source repositories of projects. A potential way of predicting faults in projects without historical data is to make use of these public data sets as training data. Cross-project fault prediction refers to predicting fault in a project using prediction models trained from historical data of other projects [7], [9]. There are some investigations concentrating on this subject and they concluded that cross-project defect prediction is still a challenging task [7], [8]. This paper reports results from the 7 PROMISE software fault data sets [10]. In this paper, after describing our data sets, we show results from learning defect predictors from design metrics. We conducted experiments after extraction of design phase metrics from 7 public projects fault data obtained from PROMISE Data Repository. We trained the learner from one projects fault data to predict the faults of other projects .We employed Naïve Bayes machine learning algorithms to construct prediction models. We found that 1) models built from design metrics is useful as they are built in early phases of the development life cycle; 2) Training data of design phase from other projects may provide better prediction results than those from the same project. Fault prediction at early stage is possible by using the information content of other projects at design phase as the training set. We therefore recommend that in future, researchers explore the effects of cross project predicator from multiple projects to identify faults in early stages of the software development life cycle. The remainder of the work is broken down as follows. Sect. II summarizes some related work; Sect. III describes the methodology of this study, including the data, learning algorithms, and the performance evaluation criteria; Section IV presents the experimental analysis. Section V provides results and discusses their implications, Section VI discusses threats to validity and Section VII concludes with a summary and future work.

Pradeep Singh is with the Department of Computer Science & Engineering, National Institute of Technology, Raipur, India (e-mail: psingh.cs@nitrr.ac.in).

Shrish Verna is with the Department of Electronics & Telecommunication Engineering, National Institute of Technology, Raipur (e-mail: shrishverma@nitrr.ac.in).

II. Related Work

Software testing is one of the most important, time consuming and critical quality assurance activities. It is a labor-intensive activity in software development life cycle while budget allocated for testing are usually limited [8], [12]. Software Testing is also the most expensive, time and resource consuming phase of the software development lifecycle requires approximately 50% of the whole project schedule [27]. 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

[13]. To this purpose, a large number of software code characteristics will be used for software fault prediction which reduces the testing time and effort which in turn reduce the overall cost of software development. In the past decade, various fault prediction models have been proposed and machine learning techniques have become more popular in constructing fault predictors [1], [11], [14], [15]. Menzies et al. [1] has evaluated the code and design attribute for fault prediction using Naïve bayes (with logNums) approach. However, most prediction models reported previously are intra-project models. The application field of these models is restricted for the projects where historical releases data is unavailable. It is also shown by NASA Researchers that a fault introduced in the requirements, which leaks into the design, code, test, integration and operational phase ensues the correction cost factor of 5 ,10, 50,130 and 368 respectively [19]. So fault prediction as early as possible can reduce the cost of correction. This research aims to extend the application field of fault prediction at design time. 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. Prior to this paper, no study was performed to investigate the relative merits of design metrics using cross project data for constructing defect predictors. However, there are few studies focusing on cross project fault prediction using static codes. Twelve real world application were used by [7] for cross-project fault predictions and they found that only few of them employed well .That means cross-project fault prediction will fail in most cases if not selecting training data carefully also cross-project fault prediction is not balanced. For example, Firefox Fault data can predict Internet Explorer faults well (Precision (76.47%) and Recall (81.25%) but the opposite direction does not work (Recall equal to 4.12%). Zimmermann et al. [7] explained that cross-project fault prediction is a serious challenge and more attention should be paid to this problem. Turhan et al. [8], [16] reported a comparative analysis between cross-company and within company data for defect prediction, to build defect predictors for local projects. They analyzed 10 projects out of which 7 were NASA projects and 3 projects from a Turkish software company. They used only static code features to build defect predictors. They concluded that cross-company data increase the probability of defect detection (pd) at the cost of increasing false positive rate (pf). Their experimental result also shows that nearest neighbor filtering can help to reduce pf when using cross-company data. However, using static code attributes from the same company is still a better choice for building defect predictors. Watanabe et al. [9] tried to build fault prediction models for inter project prediction. They trained a prediction model from a Java project and applied it to a C++ project. A method called “metrics compensation” was introduced to adapt the prediction model. One drawback of Watanabe’s work is that they only analyzed two projects in their case study, which weakens the generalization ability of their conclusions. Jureczko and Madeyski [17] tried to divide various projects into different groups by using clustering techniques. According to their definition, a group is a set of projects with similar characteristics, and a defect prediction model should work well for all projects that belong to a same group. Jureczko’s research work point out a potential way for cross-project defect prediction. If we can identify such groups successfully, we can predict defects for project without historical data by reusing defect prediction models trained from other projects belonging to the same group. Nagappan et al. [18] studied the generalization ability of prediction models for post-release defects. After investigating five Microsoft systems, they found that no single set of metrics fits all projects and predictors are accurate only when they are learned from the same or similar projects. Menzies et al. [1] also found that the best static code attributes for defect prediction vary from data set to data set, implying different data characteristics between different data sets. Different data characteristics, in addition to different contexts of projects (e.g., process, developers, programming language) make cross-project defect prediction a big challenge. Previous work has only focused on static attributes and been limited to intra project. Several authors have attempted to use cross project, but as yet at the time of writing there is still no accepted paper which tried predicting at design time using cross project.

By conducting large-scale experiments on publically available data sets, this research overcome some shortcomings of previous research work, e.g., it is hard to repeat experiments when the data used by the researches not accessible for public [7].

III. Proposed Software Fault Prediction Framework

A. Dataset Used

The data sets were obtained through the NASA Metrics Data Program, and include software measurement data and associated error data collected at the function level [22]. These software project data sets are publicly available under the PROMISE software engineering repository [10]. The types and numbers of software metrics made available are determined by the NASA Metrics Data Program. Each instance of these data sets is a program module. The quality of a module is described by its Error Rate, i.e., number of defects in the module, and Defect, whether or not the module has any defects. The latter is used as the class label. The analyses of this paper use the design code features of 7 projects tabulated in Table I. Available design codes extracted from all static features are shown in Table II. An advantage of design code features is that they can be quickly and automatically collected prior to coding, even if no other information is available. The module metrics shown in Table II have been extracted by using McCabe IQ 7.1, a reverse engineering tool that derives software quality metrics from code, visualize flow graphs and generate report documents [20]. McCabe IQ 7.1 is a reverse engineering tool that calculates metrics from flow graphs. We use the available design level module metrics. The design metrics are extracted from design phase artifacts, design

diagrams such as UML diagrams, data flow graphs and control flow graphs and. For example, Ohlsson and Alberg extract design metrics such as McCabe cyclomatic complexity from Formal Description Language (FDL) graphs in [21]. The NASA data was collected across the United States over a period of five years from numerous NASA contractors working at different geographical centers. These projects represent a wide array of projects, including satellite instrumentation, ground control systems and partial flight control modules .The data sets also represent a wide range of code reuse: some of the projects are 100% new, and some are modifications to previously deployed code. Seven projects shown in Table I are used in this study.

TABLE I
DATASETS USED IN THIS STUDY

Data	Modules	Faulty Modules	Language	Description
KC3	458	43	Java	Storage management for ground data
PC3	1563	160	C	Flight software for earth orbiting satellite
PC4	1458	178	C	Flight software for earth orbiting satellite
MW1	403	31	C	a zero gravity experiment related to combustion
MC2	161	52	C++	a video guidance system
MC1	9466	68	C\C++	a combustion experiment of a space shuttle
PC2	5589	23	C	dynamic simulator for attitude control systems

TABLE II
DESIGN METRICS USED IN THIS STUDY

Metrics	Description
Branch Count	Branch count metrics
Call Pairs	Number of calls to other functions in a module
Condition Count	Number of conditions in a module
Cyclomatic Complexity: v(g)	The number of logical independent paths v(g)=e-n+2
Decision Count	Number of decision points in a given module
Decision Density	Coundition_count/Decision_count
Design Complexity: iv(G)	The design complexity of a module
Design Density	Design density = iv(g)/v(g)
Edge Count	Number of edges found in a given module control from one module to another
Essential Complexity: ev(g)	The essential complexity of a module
Essential Density	Essential density is calculated as:(ev(g)-1/v(g)-1)
Maintenance Severity	Maintenance Severity is calculated as: :(ev(g)/v(g))
Modified Condition Count	The effect of a condition affect a decision outcome by varying that condition only
Multiple Condition Count	Number of multiple conditions that exist within a module
Node Count	Number of nodes found in a given module

We only selected 15 primitive software design metrics for our study. Design metrics are easy to extract them from design diagrams before the code becomes available. The design metrics include node_count, edge_count, and Mc- Cabe cyclomatic complexity measures which can be extracted from flowgraphs by using the McCabe IQ 7.1 tool. The static code metrics, such as num_operators, num_operands, and Halstead metrics are calculated from source codes. The other metrics are related to both the design and code. Most data sets have 15 design metrics as shown in Table II; the exception data sets are MC1 having 14 design metrics. The data sets are related to projects of various sizes written with various programming languages. Since this study focuses only on a unirepresentation approach (i.e., same design features across data sets), so only design based metrics provided by the NASA Metric Data Program are extracted. Each design metrics extracted for fault prediction and used in our study is briefly described in Table II.

Menzies et al. [1] have explored a range of data mining methods for defect prediction and found that classifiers based on Bayes theorem yields better performance than rule based methods (i.e. decision trees, oneR), for NASA data. A comparative study by Lessmann et.al also shows that Naive Bayes performs equivalently well with 15 other classifiers [2]. Therefore, Naive Bayes is a viable choice for classifier in our analysis. Naive Bayes classifiers are called "naive" because they presume independence of each feature. Naive Bayes is probabilistic classifiers and a well-known machine learning algorithm, details are skipped here.

B. Experimental Design

The pseudo code for Within Project (WP) -vs-Cross Project (CP) analysis at design level for all 7 NASA projects used in the study is given below. For each project, test sets were built from the data, selected at random. Defect predictors were then learned from:

CP data: Using each project data for model building and remaining projects for testing.

WP data: using 2/3 of the data for model building and remaining 1/3 data of that project for testing

```
DATA = [KC3, PC3, PC4, MW1, MC2, and MC1and PC2] // all available data
LEARNER = [Naive Bayes] // defect predictor
C_FEATURES <- Find common design features in DATA
FOR EACH data in DATA
data = Select C_FEATURES in data // use common design features
END
FOR EACH data in DATA
CP_TRAIN = DATA - data // cross project training data
WP_TRAIN = random 2/3% of data // within project training data
TEST = data - WP_TRAIN // 1/3 % shared test data
//construct predictor from CC data
CP_PREDICTOR = Train LEARNER with CP_TRAIN
// construct predictor from WP data
WP_PREDICTOR = Train LEARNER with WP_TRAIN
//Evaluate both predictors on the same test data
[cp_auc] = CP_PREDICTOR on TEST
[wp_auc] = WP_PREDICTOR on TEST
END
```

IV. ANALYSIS OF EXPERIMENT

This section shows the analysis of our experiments. We use Naïve Bayes classifier to evaluate the performance of a cross project design fault predicators. In this study the classifier has

four possible outcomes: If a module is fault-prone (fp) and is classified accordingly, it is counted as true positive (TP); if it is wrongly classified as not- fault- prone (nfp), it is counted as false negative (FN). Conversely, an nfp module is counted as true negative (TN) if it is classified correctly or as false positive (FP) otherwise. A large number of performance indicators which can be constructed by using these four measures explained by [22]. A good fault prediction model should identify as many fault prone modules as possible while avoids false alarms. Therefore, classifiers are predominantly evaluated by means of their TP rate (TPR), also known as rate of detection and by their FP rate (FPR) or false alarm rate [1], [23].

$$TPR = TP + (FN + TP); \quad FPR = FP/(TN + FP)$$

Note that, such metrics, although having undoubted practical value, are conceptually inappropriate for empirical comparisons of the performance of classification algorithms.

Receiver Operating Characteristic (ROC) curves compares the classification performance by a plotting the TP rate on y axis and FP rate on X axis across all the possible experiments. A typical ROC curve has a concave shape with (0, 0) as the beginning and (1, 1) as the end point. ROC curves output by two machine learner are shown in Fig. 1. The ideal point on the ROC curve would be the one when no positive examples are classified incorrectly and negative examples are classified as negative. As shown in figure the curve C1 intersects C2, determination of which model is dominating requires to calculate the area under each curve. Every model gets different values for area under curve. AUC is used to get complete order of model performance and is independent of the decision criterion selected and prior probabilities.

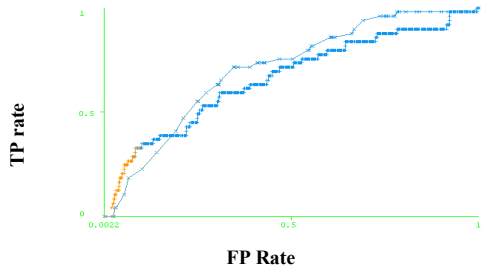

Fig. 1 ROC Curve of three experiments

AUC has also been extensively used as performance measure in other fields of research like communication (Radar), machine learning, financial and medical data mining applications. The AUC comparison can establish a dominance relationship between classifiers. The bigger the area under curve, the better the model is. As opposed to other measures, the area under the ROC curve (AUC) does not depend on the imbalance of the training set. ROC curve demonstrates several things: It shows the tradeoff between sensitivity (true positive TP) and specificity (false positive FP) and any increase in sensitivity will be accompanied by a decrease in specificity. To get fairer and more informative measure than comparing their misclassification rates area under receiver operating characteristic (AUC) is used in this study.

Higher AUC values indicate the classifier is, on average, more to the upper left region of the graph. AUC represents the most informative and commonly used, thus it is used as another performance measure in this paper. Additionally, the AUC has a clear statistical interpretation: It measures the probability that a classifier ranks a randomly chosen fp module higher than a randomly chosen nfp module, which is equivalent to the Wilcoxon test of ranks [24]. Therefore, any classifier achieving AUC well above 0.5 is demonstrably effective for identifying fault-prone modules and gives valuable advice as to which modules should receive particular attention during software testing. Comparisons are based on the area under the receiver operating characteristics curve (AUC). The merit of classifier in terms of AUC is estimated on a randomly selected hold out test set when training and testing by the same project data set, we randomly partitioned the data into training and test set using 2/3 of the data for model building and 1/3 for performance estimation. We adopt a cross testing approach to train the predictors with one project data and then after training it is applied to test the rest of the projects for fault prediction. That is, one data set is used for model building and rest of the data set is used for performance evaluation. Since we advocate using the AUC for cross project classifier comparison, the same design metric is used during model building. Most classifiers achieve promising AUC results of 0.7 and more, i.e., rank deficient modules higher than accurate ones with probability > 70 percent. Overall, this level of accuracy confirms Menzies et al.'s conclusion that "defect predictors are demonstrably useful" for identifying fp modules and guiding the assignment of testing resources [1].

V. Results

The experiments were conducted to evaluate the cross project fault prediction in early phases of software development. We used design metrics to build cross company fault prediction model. Table III presents the result of the experiment on 7 NASA MDP datasets. The bold face shows when training and testing is done with the same data set i.e model building and evaluation.

TABLE III
Prediction AUC Comparison of 7 Different Projects

	Naïve Bayes learner						
Training Set / Test Set	PC3	KC3	PC4	MW1	MC2	MC1	PC2
PC3	**0.718**	0.68	0.588	0.684	0.681	0.691	0.694
KC3	0.792	**0.77**	0.624	0.801	0.813	0.816	0.81
PC4	0.743	0.668	**0.755**	0.695	0.638	0.634	0.646
MW1	0.743	0.739	0.511	**0.785**	0.749	0.756	0.746
MC2	0.526	0.683	0.375	0.661	**0.709**	0.699	0.684
MC1	0.781	0.843	0.71	0.541	0.852	**0.794**	0.833
PC2	0.715	0.777	0.649	0.772	0.773	0.815	**0.693**

Column 2 shows the result of experiment when training is done by PC2-1563 dataset and prediction performance were evaluated for rest of the data sets. As our results indicate that

out of six cross project data, KC3,PC4,MW1,MC1 i.e. 4 projects has given better result than within project. Also all these four cross projects achieved promising AUC results of more than 0.7. This means that the performance of fault prediction based on design metric improves when cross project modules are evaluated. From Table III it can be seen that almost every cross project design level prediction achieving AUC well above 0.5 which is effective for identifying fault-prone modules [2] and can give valuable advice to designer of the software about that which modules should receive particular attention. For all the result, we noticed that max value AUC is 0.852 which is achieved during cross project prediction and the max value of within project AUC is 0.785. As our results indicate that most of the result of cross project are competitive and value of AUC > 0.5. It indicates that the cross projects fault prediction using design metrics are surprisingly good at fault proneness prediction.

VI. Threats to Validity

When conducting an empirical study, it is important to be aware of potential threats to the validity of the obtained results and derived conclusions. Threats to generalized the results are bias relates to the data used, e.g., its measurement, tools used for measurement, developers efficiency accuracy and representativeness. Using public domain data secures the results in so far as that they can be verified by replication and compared with findings from previous experiments. Also, several authors have argued in favor of the appropriateness and representativeness of the NASA MDP repository and/or used some of its data sets for their experiments [1], [26]. Therefore, we are confident that the obtained results are relevant for the software defect prediction community. To be trustworthy the software engineering community stipulates that the subject of an empirical study have the following characteristics.

The software project should be developed by a group, large enough as industry-size projects, and not a dummy problem. It should be developed by software professionals and not by students, developed in an industry/government organization setting, and not in labs [25].

We note that our case studies fulfill all of the above criteria. The software systems investigated in our study were developed by professionals in a government software development organization. In addition, each system was developed to address a real-world problem. Empirical studies that evaluate measurements and models across multiple projects should take care in assessing the scope and impact of its analysis and conclusion.

VII. Conclusions

In this paper, we have reported an empirical study for cross project fault prediction at design phase over 7 public domain software development data sets from the NASA MDP repository. The AUC was recommended as the primary accuracy indicator for comparative studies in software fault prediction since it separates predictive performance from class and cost distributions. The overall predictive accuracy across all cross project are confirmed the general appropriateness of fault prediction using at design level to identify fault prone software modules (with AUC > 0.5) and guide the assignment of testing resources. The experiment we have conducted revels that design metrics, from software early lifecycle are good predictors for software faulty modules. Thus, we conclude that metrics from the early software lifecycle are useful and should be used. Regardless of the data from within project, software fault prediction model can be built at design phase with other project's design level fault data.

Acknowledgment

This study is supported by The Scientific and Technological Council of C.G. under Grant 8068/CCOST. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

References

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

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

[3] C. Andersson, "A Replicated Empirical Study of a Selection Method for Software Reliability Growth Models," *Empirical Software Eng*., vol. 12, no. 2, pp. 161-182, 2007.

[4] N. E. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Trans. Software Eng*., vol. 26, no. 8, pp. 797-814, Aug. 2000.

[5] Tosun, A., Turhan, B., Bener, "A.: Practical considerations in deploying AI for defect prediction: a case study within the Turkish telecommunication industry." *In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pp. 1–9 (2009).

[6] Weyuker, E. J., Ostrand, T. J., Bell, R. M. "Comparing the effectiveness of several modeling methods for fault prediction". *Empir. Softw. Eng*. 15(3), 277–295 (2009)

[7] Zimmermann, T., Nagappan, N., Gall, H.: "Cross-project defect 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)

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

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

[10] http://promisedata.org/repository.

[11] 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)

[12] 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)

[13] 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)

[14] 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)

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

[16] Turhan, B., Bener, A., Menzies, T. "Regularities in learning defect predictor," *In: The 11th International Conference on Product Focused Software Development and Process Improvement,* pp. 116–130 (2010)
[17] Jureczko, M., Madeyski, L. "Towards identifying software project clusters with regard to defect prediction," *In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering,* pp. 1–10 (2010)
[18] Nagappan, N., Ball, T., Zeller, A. "Mining metrics to predict component failure" In: *Proceedings of the 28th International Conference on Software Engineering*, pp. 452–461 (2006).
[19] http://www.cse.lehigh.edu/~gtan/bug/localCopies/nistReport.pdf
[20] Do-178b and mccabe iq. Available in http://www. mccabe.com/iq_research_whitepapers.htm.
[21] N. Ohlsson and H. Alberg "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.
[22] K. El-Emam, S. Benlarbi, N. Goel, and S.N. Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High-Risk Software Components," *J. Systems and Software*, vol. 55, no. 3, pp. 301-320, 2001.
[23] T. M. Khoshgoftaar and N. Seliya, "Analogy-Based Practical Classification Rules for Software Quality Estimation," *Empirical Software Eng.,* vol. 8, no. 4, pp. 325-350, 2003.
[24] T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, 2006.
[25] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.
[26] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust Prediction of Fault-Proneness by Random Forests," Proc. 15th *Int'l Symp Software Reliability Eng.*, 2004.
[27] M.J. Harrold, Testing: a roadmap, in: *Proceedings of the Conference on the Future of Software Engineering, ACM Press, New York*, NY, 2000.