A Two-Step Model for Defect Density Estimation

Onur Kutlubay, Burak Turhan, Ayşe B. Bener
Department of Computer Engineering, Boğaziçi University
34342, Bebek, Istanbul, Turkey
kutlubay@cmpe.boun.edu.tr, {turhanb, bener}@boun.edu.tr

Abstract

Identifying and locating defects in software projects is a difficult task. Further, estimating the density of defects is more difficult. Measuring software in a continuous and disciplined manner brings many advantages such as accurate estimation of project costs and schedules, and improving product and process qualities. Detailed analysis of software metric data gives significant clues about the locations and magnitude of possible defects in a program.

The aim of this research is to establish an improved method for predicting software quality via identifying the defect density of fault prone modules using machine-learning techniques.

We constructed a two-step model that predicts defect density by taking module metric data into consideration. Our proposed model utilizes classification and regression type learning methods consecutively. The results of the experiments on public data sets show that the two-step model enhances the overall performance measures as compared to applying only regression methods.

1. Introduction

The Standish Group carried out a survey in nearly 10-000 projects that took place in mostly US and European companies [4]. The survey results show that 53% of the software projects are challenged (i.e., late, over budget and/ or with less than the required features and functions); 18% have failed (i.e., cancelled prior to completion or delivered and never used); and only 29% succeeded. These statistics show the importance of measuring software projects early in their life cycle and taking necessary actions to mitigate the risks. When software projects are carried out in the industry, an extensive metrics program is usually seen unnecessary, and software project managers start to stress on putting together a metrics program when things are bad or when there is a need to satisfy some external assessment body [6]. Once a metrics program is established, there are publicly available well-established tools for extracting metrics such as size, McCabe's cyclomatic complexity, and Halstead's program vocabulary [17], [10]. These tools help automating the data collection process in software projects.

Models for assessing software risk in terms of predicting defects in a specific module or function have been proposed in previous research [7]. Some recent models also utilize machine-learning techniques for defect prediction [3], [25]. Testing is the most popular method for defect detection in most of the software projects. However, when the size of projects grow in terms of both lines of code and effort spent, the task of testing gets more difficult and computationally expensive with the use of sophisticated testing and evaluation procedures. Therefore, research in this area has focused on finding cost effective methods in defect detection and prediction [7], [27], [28].

Popular ways of detecting software defects prior to testing phase are expert judgments and manual code reviews. Although code reviews are accepted to be an effective solution they are labor intensive [22]. Therefore there is a need to find less costly methods which are as good as manual code reviews.

The software metric data gives us the values for specific variables that measure a module/ function or the entire software. This data can be used as an input for a machine learning system together with the weighted error/ defect data of the corresponding modules/ functions. A learning system is defined as a system that is said to learn from experience with respect to some class of tasks and performance measure, such that its performance at these tasks improves with experience [23].

We claim that we can obtain better defect prediction results with a two step model, and the motivation behind this will become clear in the following sections. Such a model suggests that we should first classify modules as defective and defect-free; second we should take defective data that is identified in the first step and carry out a regression analysis on this data set which contains the modules that are identified as defective. As a result, our proposed model gives an estimation of the defect densities in the defective modules. The

33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)
0-7695-2977-1/07 $25.00 © 2007 IEEE

model helps us to concentrate on specific modules that contain more defects. Therefore, we can save significant amount of time and resources.

This paper is organized as follows: The second section gives a brief literature survey on previous research. The third section states the problem and the fourth section explains the details of the two-step model and methods used in our experiments. In the fifth section, we have listed the results of the experiments and a detailed evaluation of the machine learning algorithms. The last section concludes our work and discusses the future research that could be done in this area.

2. Related Work

Defect prediction models differ mainly for two reasons. First, models may use different types of metrics. Most of the defect prediction models use the basic metrics such as complexity and size of the software [11]. However, in some cases, researchers came up with more advanced metrics. For instance, Cusumano developed testing metrics to determine the sequence of defects [5]. Second, different stages of the software development process may have different requirements. A quality assessment model that is successful in the design phase is probably not so in the development or test phases [12].

Machine learning algorithms have been proven to be practical in poorly understood problem domains that have changing conditions [28]. Software quality problems can be formulated as learning processes. Therefore, it would be possible to apply regular machine learning algorithms to come up with estimations for defect analysis [7], [28]. Decision trees, artificial neural networks, Bayesian belief networks and classification techniques such as k-nearest neighbor are examples of most commonly used techniques for software defect prediction problems [3], [9], [13], [14], [23], [28].

Performance measures like probability of detection (PD), probability of false alarm (PF), accuracy and precision are among the most common methods for evaluating the success of the previously proposed defect detectors [18]. Menzies et.al. argue that another important performance criteria for a defect detector is effort [21]. Effort constitutes to the amount of resource that should be allocated to V&V tasks on the modules predicted as defective. In their research they evaluate several statistical, human expert and machine-learning based techniques against the same data set in terms of these performance metrics including the effort [22].

Machine learning is also used to generate models of program properties that are known to cause errors. Support vector machines and decision tree learning tools are implemented to classify and investigate the most relevant subsets of program properties [2]. Underlying intuition is that most of the properties leading to faulty conditions can be classified within a few groups. Fault relevant properties are utilized to generate a model, and this pre-computed function selects the properties that are most likely to cause errors and defects in the software.

In their research, Podgurski et al. used clustering over function call profiles to determine which features enable a model to distinguish failures from non-failures [26]. The researchers used a technique called dynamic invariant detection to pick possible invariants from a test suite and investigate violations that usually indicate an erroneous state. This method is also used to determine counter examples and to find properties that lead to correct results for all conditions [8].

Previous research usually deals with the characteristics of the detectors or estimators in defect prediction problem domains. Koru and Liu in their research argue that, besides the characteristics of the detector itself, another important property of successful models is how they handle the input data. More specifically; they claim that, organizing the input data set at fine granularity level (i.e. class level abstraction against method level) would bring out better results in defect prediction [15].

The previous research tackled the problem of software defect prediction by either using classification or regression type methods. This has inspired us to combine classification and regression type of prediction methods to find fault prone modules in software programs.

3. Problem statement

Two types of research can be performed on the static code metrics in terms of defect prediction [25]. The first one is predicting whether a given code segment is defective or not. The second one is predicting the magnitude of the possible defect, if any, with respect to various viewpoints such as density, severity or priority.

There are many doubts on the usefulness of binary defect prediction models based on static code metrics. The reason is mainly threefold. First they are usually collected from modules without knowing how often the module will be called. Second the severity of the problem is not known. Third the interconnections among the modules are unknown. Therefore binary

33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)
0-7695-2977-1/07 $25.00 © 2007 IEEE

defect prediction models based on static code metrics are not accepted as replacement for standard practices in which test engineers use their domain knowledge and available documentation to decide where to allocate the scarce resources [18], [19]. Hence, the modules detected by the defect detectors based on static code metrics deserve attention along with the ones that are identified by the test engineers. As Menzies et.al. described, these defect detectors are considered as secondary tools in defect prediction process [21]. If we take this one step forward, we could argue that when the defect prediction model generates numerically comparable results rather than just binary classification secondary detectors give precedence information as well. Additionally, test engineers may benefit even more if the defect predictions include properties such as severity and criticality, rather than just defect information. Therefore, our work in this research is primarily focused on predicting the magnitude of the possible defects. Rather than trying to find out which modules are fault prone, we specifically stress on determining the possible defect densities of the modules. By doing so, we aim to provide the software quality practitioner with an estimation about "which modules may contain more faults". This information can be used to assign priorities to modules that are predicted as defective, and helps to allocate the scarce testing and validation resources to the modules that are predicted as "most defective".

4. Two-Step Model and Methodology

Data sets used in the experiments are from NASA Metric Data Program [24] and cover a wide spectrum of project sizes. The defect percentages of the datasets also differ. For example, there is a difference between the defect percentages of the projects PC2 and JM1 so that, even if both projects have large number of modules, PC2 has very few defects. We believe that using data sets with different properties help us to validate our proposed model. The data sets include mostly defect-free modules, so there is a bias towards underestimating the defect density in the prediction process. Also it is likely that any other input data set will have the same characteristic since it practically would have much more defect-free modules than defective ones in real life software projects. In order to remove this bias, we can divide the defect prediction problem into two parts. The first part consists of predicting whether a given module is defective or not. And the second part is predicting the magnitude of the possible defect if it is labeled as defective by the first part. If we enhance the classification step of the model, we would be able to improve the overall performance of the learning system.

4.1. Performance Evaluation

Our analysis depends on machine learning techniques. As a widely known data mining standard, we used 10-fold cross validation in our experiments. After training a learner with the training data set, the learned theory should be tested on data that is not used to build it [18], [20]. In 10-fold cross validation, the data set is divided randomly into 10 equal sized parts. To generate each training/ validation data set pair, we keep one of the 10 parts out as the validation set, and combine the remaining 9 parts to form the training set. Doing this 10 times, each leaving out another one of the 10 parts out, we get 10 pairs [1].

Even though our main goal was to estimate the defect densities of the modules, we initially utilized classification algorithms to assess their effectiveness in predicting whether a module is defective or not. The results of these experiments and similar research in the literature inspired us to use the outputs of such detectors to regression-type learners which try to estimate the defect density values [15], [22]. The results of classification experiments which include 2-classes can be assessed using a "confusion matrix" which is depicted in Table 1 [1].

Table 1. Confusion Matrix

	Predicted Class	
True Class	Yes	No
Yes	TP: True Positive	FN:False Negative
No	FP: False Positive	TN: True Negative

According to this confusion matrix, incorrect predictions reside in two regions. The system may predict that a defect-free module as defective which is shown as FP region (a.k.a. false alarm) or the system may not identify an actually defective module which is shown as FN region (a.k.a. miss). The other two regions mean that the predictions are correct. In most cases, FP and FN regions have different costs for the project owners [20]. Some real life projects that are mission critical cannot even tolerate a single miss. In this case, the cost of an unidentified fault is higher than the cost of erroneously predicting a module as defective. In contrast, some projects may have very tight testing budgets so that classifying a defect-free module as defective results in wasting the resources. One can employ different methods for classification and try to fine tune the learning model. In this respect,

the detectors can be evaluated based on receiver operator characteristic (ROC) curves.

Again relying on the confusion matrix a fine set of assessing the performance of classification algorithms consists of Accuracy (Acc), Probability of False Alarm (PF) and Probability of Detection (PD) values. We have used these metrics in our experiments to evaluate the performances of the classification. For evaluating the performance of regression-type prediction models, we have used Root Mean Square Error (RMSE) measure.

In our classification experiments we used Naive Bayes and Decision Tree techniques, and in regression experiments we used Radial Basis Function (RBF) and Regression Tree techniques.

Naive Bayes classifiers often work much better in many complex real-world situations than might be expected. Also in problem domains like our, they are widely used efficiently [18].

Decision trees are one of the most popular approaches for both classification and regression type predictions [23]. They are generated based on specific rules. In the regression type prediction experiments, we used regression trees which may be considered as a variant of decision trees, designed to approximate real-valued functions instead of being used for classification tasks.

Radial Basis Function is a variant of Artificial Neural Network (ANN). RBF networks typically have two distinct layers. The bottom layer consists of a set of basis functions. The top layer is a simple linear discriminant that outputs a weighted sum of the basis functions [16]. Gaussian Hidden Unit functions are used as the basis functions in our experiments.

In our research, we carried out each type of experiments by utilizing two different machine learning techniques. By doing this, we try to see to what extend our experiments can be generalized. The algorithms we use are selected among the simplest techniques in this manner. Another goal is to compare these algorithms' performances. We used t-tests to decide if the results are statistically different with a 0.05 significance level and evaluated the results by comparing their corresponding error rate/ accuracy values. Moreover, we enhance the regression experiments with a second step which relies on the results of the classification experiments. So the results of these two regression experiments are compared using the same method.

5. Experiments and Results

In the first type of experiments, our aim was predicting whether a module is defective or not. Table 2 lists the Accuracy, PD, PF test results from a 10 x 10-fold cross validation experiment for each of the data sets. For each of the data sets, the experiment results for Decision Tree and Naive Bayes techniques are compared using t-tests to see whether they are statistically different; and the winner is listed if they are statistically different.

The prediction accuracies are between 62% and 95% for all of the projects and the PD values are between 7% and 62%. Decision Tree technique wins for six data sets in terms of accuracy where there is a tie for two of the data sets. Naive Bayes wins only for PC3 data set. In contrast with the accuracy results, Naive Bayes wins five of the nine test cases in PD values where Decision Tree wins for only two of them.

Table 2. Classification results

		Accuracy	PD	PF
JM1	Dec.Tree	76.07	35.13	13.94
	N.Bayes	62.31	45.86	24.82
	Winner	Dec.Tree	N.Bayes	N.Bayes
CM1	Dec.Tree	86.12	22.36	7.27
	N.Bayes	81.60	25.40	13.28
	Winner	Dec.Tree	Tie	N.Bayes
KC1	Dec.Tree	85.03	33.01	7.43
	N.Bayes	72.43	62.32	33.58
	Winner	Dec.Tree	N.Bayes	N.Bayes
KC3	Dec.Tree	87.07	30.99	7.21
	N.Bayes	80.25	40.06	17.27
	Winner	Dec.Tree	N.Bayes	N.Bayes
MW 1	Dec.Tree	90.09	25.14	4.86
	N.Bayes	89.14	11.80	4.44
	Winner	Tie	Dec.Tree	Tie
PC1	Dec.Tree	91.34	40.38	4.94
	N.Bayes	79.80	46.03	20.65
	Winner	Dec.Tree	Tie	N.Bayes
PC2	Dec.Tree	95.20	7.76	1.36
	N.Bayes	94.16	17.40	2.52
	Winner	Tie	N.Bayes	N.Bayes
PC3	Dec.Tree	86.49	28.47	7.43
	N.Bayes	88.69	8.81	2.84
	Winner	N.Bayes	Dec.Tree	Dec.Tree
PC4	Dec.Tree	88.72	42.84	6.29
	N.Bayes	85.15	47.10	8.34
	Winner	Dec.Tree	N.Bayes	N.Bayes

As a result, Naive Bayes is more successful in defect detection but its accuracy is not so. This is because Naive Bayes also has high PF values. Similar to other examples in the literature, we found that both techniques, especially Naive Bayes, perform as good as manual inspections or expert judgments in terms of PD values [18].

Table 3. Regression results on the entire data sets

		RMSE
JM1	Reg.Tree	37.81
	RBF	55.92
	Winner	Reg.Tree
CM1	Reg.Tree	24.36
	RBF	31.04
	Winner	Reg.Tree
KC1	Reg.Tree	46.94
	RBF	53.22
	Winner	Reg.Tree
KC3	Reg.Tree	59.31
	RBF	72.58
	Winner	Reg.Tree
MW1	Reg.Tree	14.87
	RBF	22.76
	Winner	Reg.Tree
PC1	Reg.Tree	34.69
	RBF	40.04
	Winner	Reg.Tree
PC2	Reg.Tree	15.75
	RBF	21.22
	Winner	Reg.Tree
PC3	Reg.Tree	45.46
	RBF	51.53
	Winner	Reg.Tree
PC4	Reg.Tree	39.08
	RBF	48.15
	Winner	Reg.Tree

As we move on to our primary goal which is predicting the defect density values we first started with experiments on the entire data sets one by one again using a 10 x 10-fold cross validation method. After that we evaluated the resulting RMSE values using t-tests. The experiment results generally showed us that, applying regression techniques directly on the data set which contains both defective and defect-free modules results to high RMSE values. Table 3 depicts the experiment results for each data set and the winners as a result of the t-tests. The results are far from being acceptable since both of the method fails to approximate the defect density values well. In these experiments, even both of the techniques' results are not very good; Decision Trees are definitely more successful since it wins on all test cases. Doing a regression type learning on the entire data set to estimate the defect density values is probably the most straightforward method. The RMSE values range from 14.87 to 72.58 depending on the data set and the machine learning technique used. Even though we claim that these values are high, this is just our observation and depends on intuition. Therefore we need another set of experiment results to validate our observation and to show that these estimations can be improved.

Keeping the classification results in mind, and considering our comments on the first regression results we decided to do another set of experiments using the same regression techniques. The high error rates on the previous regression experiments were partly due to the nature of the data sets, that is, they contain much more defect-free modules than defective ones so the predictor had a bias towards underestimating the defect densities. Therefore, the same regression experiment using the modules that are identified as defective by the classification techniques can overcome this problem. Besides, the overall accuracy will not depend solely on the regression performance but the error rates of the classification step plus the regression on the "defect-prone" modules. Assuming that the regression experiments will yield better accuracies over the data set containing only defective items, by using a better classifier, or fine tuning the classifier for different aspects of the project, the overall prediction accuracy will improve a lot. In other words, the problem is reduced to only predicting whether a module is defective or not. In our experiments, we used Naive Bayes technique in the classification step because it outperformed Decision Tree technique in most of the test cases earlier.

The experiment results are collected in two ways, RMSE values of the regression step alone, and the overall RMSE values (including misses and false alarms in the classification step). The calculation of the RMSE depends on where classification results reside in the confusion matrix.

The experiment results for each project are listed in Table 4. The first of the three RMSE columns stand for the values related to the regression step alone (without taking the errors originating from the misclassified modules into account). Second RMSE column shows the calculated overall RMSE values as described above. And the last column shows the RMSE results of

Table 4. Performance of the two-step model

		RMSE for Step 2	Overall RMSE		Old RMSE (Recall Table 3)
JM1	Reg.Tree	20.03	**27.40**	WIN	37.81
	RBF	24.52	33.34	WIN	55.92
	Winner	Reg.Tree	Reg.Tree		Reg.Tree
CM1	Reg.Tree	14.42	**21.32**	WIN	24.36
	RBF	19.81	25.20	WIN	31.04
	Winner	Reg.Tree	Reg.Tree		Reg.Tree
KC1	Reg.Tree	20.01	**31.42**	WIN	46.94
	RBF	19.34	**30.60**	WIN	53.22
	Winner	Tie	Tie		Reg.Tree
KC3	Reg.Tree	44.43	**47.25**	WIN	59.31
	RBF	55.86	60.11	WIN	72.58
	Winner	Reg.Tree	Reg.Tree		Reg.Tree
MW1	Reg.Tree	10.81	**13.64**	WIN	14.87
	RBF	10.81	**13.64**	WIN	22.76
	Winner	Tie	Tie		Reg.Tree
PC1	Reg.Tree	16.35	**23.43**	WIN	34.69
	RBF	24.50	28.75	WIN	40.04
	Winner	Reg.Tree	Reg.Tree		Reg.Tree
PC2	Reg.Tree	4.76	**5.40**	WIN	15.75
	RBF	8.13	8.66	WIN	21.22
	Winner	Reg.Tree	Reg.Tree		Reg.Tree
PC3	Reg.Tree	23.04	**38.57**	WIN	45.46
	RBF	26.20	**39.98**	WIN	51.53
	Winner	Tie	Tie		Reg.Tree
PC4	Reg.Tree	23.72	**27.51**	WIN	39.08
	RBF	28.29	31.86	WIN	48.15
	Winner	Reg.Tree	Reg.Tree		Reg.Tree

the experiments that are done over the entire data sets (i.e. experiment results in Table 3).

For all project data sets, the regression step alone is successful. However, this evaluation would have been a measure of overall success only if the classification step revealed the really defective modules and nothing else. So the real evaluation must be done according to the overall RMSE values. When the overall RMSE values are compared to the previous RMSE values which resulted from single step regression experiments; for all of the data sets, both techniques win. A comparison between the two techniques shows that Regression Tree again outperforms RBF since it wins for six data sets where the remaining three values are tied according to the t-tests. These results show that the two-step model enhances the performance of the regression.

5.1 Threats to Validity

We have evaluated our model on several public datasets from NASA, which are real software projects and acknowledged to reflect the general properties of software industry in previous research [18]. Defect rates of these projects also validate our motivation for constructing the two-step model (i.e. most of the software modules are non-defective). In the experimental setup a 10x10 cross validation framework is employed, which is accepted as a standard in many data mining research. In each fold the order of data is randomized in order to remove any ordering effect. The methods used for classification and regression are chosen among the best performing ones as reported in previous research. Among these methods, RBF may suffer the problem of over fitting and local minima since it is an iterative learning algorithm. We have

taken necessary precautions to avoid this problem by fine tuning its learning parameters and checking the epoch vs. test error plots.

6. Conclusions

The goal of a software engineer is to develop a software program that produces the desired results on time and within budget. This involves minimizing the defects in the programs. In order to improve software quality, high risk components in the software project should be caught as early as possible. In this research we proposed an enhanced defect density prediction model based on machine learning techniques. We believe that software practitioners can reduce the costs introduced by the labor dependent manual code reviews by using such models.

Our proposed model is a two step model. It takes historical software metric data as its input. In real life projects, we see that a data set contains more defect free modules than defective ones. Machine learning methods fail to predict defect densities accurately when regression is applied in such data sets. However, our model first classifies the data set as defective and defect free, and then, it applies regression. Our experiment results showed that classification is a better approach when the data set has more defect free modules than defective ones.

Our major contribution is to combine classification and regression methods in software defect prediction. Along with predicting defective modules, our proposed model also generates estimations on the defect densities. Therefore software quality professionals may use the model to better allocate scarce resources in testing.

As a future work, different machine learning algorithms or the improved versions of the ones we used may be included in the experiments. The two steps of the model can be thought of separately and combinations of different learners may be applied in these steps. Our model may also be extended to consider or to differentiate specific costs of misclassifying a defective module and misclassifying a non-defective module. Furthermore, our models performance can be compared with expert judgments performance. We also believe that the proposed model may be used in different problem domains, which show similar characteristics to that of software quality.

Acknowledgements

This research is supported in part by Bogazici University research fund under grant number BAP-06HA104.

References

[1] Alpaydin, E., *Introduction to Machine Learning*, The MIT Press, 2004.

[2] Brun, Y., and Ernst, M. D., "Finding latent code errors via machine learning over program executions", *Proceedings of the 26th International Conference on Software Engineering*, 2004.

[3] Ceylan, E., Kutlubay, O., Bener, A., "Software Defect Identification Using Machine Learning Techniques", *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, 2006, pp. 240 – 247.

[4] The Standish Group, *CHAOS Chronicles - 2004 Third Quarter Research Report*, 2004.

[5] Cusumano, M.A., *Japan's Software Factories*, Oxford University Press, 1991.

[6] Fenton, N., and Neil, M., "Software metrics: roadmap", *Proceedings of the Conference on The Future of Software Engineering, International Conference on Software Engineering*, 2000, pp. 357-370.

[7] Fenton, N., and Neil, M., "A critique of software defect prediction models", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, 1999, pp. 675-689.

[8] Groce, and Visser, W., "What went wrong: Explaining Counterexamples", *10th International SPIN Workshop on Model Checking of Software*, Portland, Oregon, 2003, pp. 121–135.

[9] Jensen, F.V., *An Introduction to Bayesian Networks*, Springer, 1996

[10] Halstead, M., *Elements of Software Science*, Elsevier. 1977

[11] Henry, S., and Kafura, D., "The Evaluation of Software System's Structure Using Quantitative

Software Metrics", *Software Practice and Experience*, vol. 14, no. 6, 1984, pp. 561-573.

[12] Hudepohl, P., Khoshgoftaar, M., Mayrand, J., "Integrating Metrics and Models for Software Risk Assessment", *The Seventh International Symposium on Software Reliability Engineering (ISSRE '96),* 1996

[13] Khoshgoftaar, T.M., Allen, E.B., "A Comparative Study of Ordering and Classification of Fault-Prone Software Modules", *Empirical Software Engineering*, 4, 1999, pp. 159–186.

[14] Khoshgoftaar, T.M., Allen, E.B., "Classification of Fault-Prone Software Modules: Prior Probabilities, Costs, and Model Evaluation", *Empirical Software Engineering*, 3, 1998, pp. 275–298.

[15] Koru, G., and Liu, H., "Building Effective Defect-Prediction Models in Practice", *IEEE Software*, Vol.22, No. 6, 2005.

[16] Li, S. T., "Robust Radial Basis Function Networks", *Institute of Information Management* National Cheng Kung University, 2003

[17] McCabe, T., "A complexity measuree", *IEEE Transactions on Software Engineering*, 2(4), 1976, pp. 308-320.

[18] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors", *IEEE Transactions on Software Engineering*, 33(1), 2007, pp. 2–13.

[19] Menzies, T., Di Stefano, J.S., Cunanan, C., Chapman, R., "Mining repositories to assist in project planning and resource allocation", *26th International Conference on Software Engineering,* 2004, pp. 75 – 79.

[20] Menzies, T., Di Stefano, J.S., "How good is your blind spot sampling policy", *Proceedings. Eighth IEEE International Symposium on Software Metrics*, 2004.

[21] Menzies, T., DiStefano, J., Orrego, A., Chapman, R, "Assessing Predictors of Software Defects", *Proceedings, workshop on Predictive Software Models*, 2004

[22] Menzies, T., Di Stefano, J.S., Ammar, K., McGill, K., Callis, P., Chapman, R.M., Davis, J., "When Can We Test Less?", *Proceedings of the 9th International Symposium on Software Metrics*, 2003

[23] Mitchell, T.M., *Machine Learning*, McGrawHill, 1997

[24] NASA/WVU IV&V Facility, Metrics Data Program, available from http://mdp.ivv.nasa.gov.

[25] Neumann, D.E., "An Enhanced Neural Network Technique for Software Risk Analysis", *IEEE Transactions on Software Engineering*, 2002, pp. 904-912.

[26] Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B.Wang, "Automated support for classifying software failure reports", *International Conference on Software Engineering*, 2003, pp. 465–475.

[27] Porter, A.A., Selby, R.W., "Empirically Guided Software Development Using Metric-Based Classification Trees", *IEEE Software*, 1990.

[28] Zhang, D., "Applying Machine Learning Algorithms in Software Development", *The Proceedings of 2000 Monterey Workshop on Modeling Software System Structures*, Santa Margherita Ligure, Italy, 2000, pp.275-285.