Static Analysis Tools as Early Indicators of Pre-Release Defect Density

Nachiappan Nagappan[+]
Department of Computer Science
North Carolina State University
Raleigh, NC 27606
nnagapp@ncsu.edu

Thomas Ball
Microsoft Research
Redmond, WA 98052
tball@microsoft.com

ABSTRACT
During software development it is helpful to obtain early estimates of the defect density of software components. Such estimates identify fault-prone areas of code requiring further testing. We present an empirical approach for the early prediction of pre-release defect density based on the defects found using static analysis tools. The defects identified by two different static analysis tools are used to fit and predict the actual pre-release defect density for Windows Server 2003. We show that there exists a strong positive correlation between the static analysis defect density and the pre-release defect density determined by testing. Further, the predicted pre-release defect density and the actual pre-release defect density are strongly correlated at a high degree of statistical significance. Discriminant analysis shows that the results of static analysis tools can be used to separate high and low quality components with an overall classification rate of 82.91%.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Symbolic execution, Testing tools.* D.2.8 [**Software Engineering**]: Metrics – *Performance measures, Process metrics, Product metrics.*

General Terms
Measurement, Reliability, Languages.

Keywords
Static analysis tools, Defect density, Statistical methods, Fault-proneness.

1. INTRODUCTION
Static analysis explores all possible execution paths in a program at compile time. Static analysis tools for finding low-level programming errors are becoming more commonplace [6, 10, 11]. Such tools identify errors such as buffer overflows, null pointer dereferences, the use of uninitialized variables, etc. Static analysis can find errors that occur on paths uncovered by testing. However, static analysis may produce false errors.

Testing, on the other hand, has the ability to discover not only the "shallow" errors exposed by static analysis tools but also to expose deep functional and design errors. We cannot expect static analysis tools to find such errors. Said another way, programmers make many different kinds of errors. Static analysis tools find certain classes of these errors while testing must be used to find the other classes of errors.

It is beneficial to obtain early estimates of system reliability or fault-proneness to help inform decisions on testing, code inspections, design rework, as well as financial costs associated with a delayed release, etc. In industry, estimates of system reliability (or pre-release defect density) are often available too late to affordably guide corrective actions to the quality of the software. Several studies have been performed to build models that predict system reliability and fault-proneness.

Static analysis tools are deployed early in the software development process, either on the developer's desktop computer or during nightly builds of the product. Typically, static analysis tools inspect the code before it is ever tested by a human. Let the *static analysis defect density* of a software component be the number of defects found by static analysis tools per KLOC (thousand lines of code). Let the pre-*release defect density* of a software component be the number of defects per KLOC found by other methods, before the component is released.

Our basic question is whether or not we can use static analysis defect density as a predictor of pre-release defect density. That is, are static analysis tools leading indicators of faulty code? The hypotheses that we address in this paper are:

- static analysis defect density can be used as an early indicator of pre-release defect density;
- static analysis defect density can be used to predict pre-release defect density at statistically significant levels;
- static analysis defect density can be used to discriminate between components of high and low quality (fault and not fault-prone components)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE '05, May 15–21, 2005, St. Louis, MO, USA.
Copyright 2005 ACM 1-58113-963-2/05/0005...$5.00.

[+] Nachiappan Nagappan was an intern with the Testing, Verification and Measurement Group, Microsoft Research in the summer of 2004 when this work was carried out.

We analyze these hypotheses inside Microsoft, where the PREfix and PREfast static analysis tools have been widely deployed. The PREfix tool finds common programming errors using symbolic execution, applied bottom-up over the call graph of a program. PREfix is run regularly over the entire Windows source code and the defects it finds are automatically entered into a defect database and assigned to programmers to fix. PREfix typically runs as part of the centralized build of Windows. PREfast is a lightweight version of PREfix that can be run by developers on their desktop machines. It performs a local dataflow analysis on each function to find common programming errors.

Because of the wide and automated use of PREfix and PREfast within the Windows organization, as well as the use of defect tracking databases, we are in an excellent position to assess the above hypotheses on a large amount of data.

Our results show that the static analysis defect density is correlated at statistically significant levels to the pre-release defect density determined by various testing activities. Further, the static analysis defect density can be used to predict the pre-release defect density with a high degree of sensitivity. Discriminant analysis performed using static analysis defect density as the dependent variable results in an overall classification rate of 82.91 % with respect to identifying fault and not fault-prone components.

The organization of the paper is as follows. Section 2 reviews related work. Section 3 introduces the PREfix and PREfast static analysis tools and explains how these tools fit into the development process at Microsoft. Section 4 presents our case study. Section 5 concludes and discusses future work.

2. RELATED WORK

Fault-proneness is defined as the probability of the presence of faults in the software [8]. Research on fault-proneness has focused on two areas: (1) the definition of metrics to capture software complexity and testing thoroughness and (2) the identification of and experimentation with models that relate software metrics to fault-proneness [8]. Fault-proneness can be estimated based on directly-measurable software attributes if associations can be established between these attributes and the system fault-proneness [21].

Structural object-oriented (O-O) measurements, such as those in the CK O-O metric suite [7], have been used to evaluate and predict fault-proneness [1, 4, 5]. These metrics can be a useful early internal indicators of externally-visible product quality [1, 25, 26].

A number of techniques have been used for the analysis of software quality. For example, multiple linear regression analysis was used to model the dependence of quality on software metrics [18, 22]. The multiple coefficient of determination, R^2, provides a quantification of how much variability in the quality can be explained by the regression model. One difficulty associated with multiple linear regression is multicollinearity among the metrics, which can lead to inflated variance in the estimation of reliability.

One approach which has been used to overcome this is Principal Component Analysis (PCA) [12]. In PCA a smaller number of uncorrelated linear combinations of metrics, which account for as much sample variance as possible, are selected for use in regression (linear or logistic). PCA can be used to select a subset of the metrics (principal components) that maximize variance of the controlling (independent) variable, such as reliability. PCA can also be used to remove metrics that are highly correlated with other metrics, thus simplifying data collection with minimal impact on the accuracy of the information provided. A multivariate logistic regression equation [9] can be built to model the data, using the principal components as the variables. Denaro et al. [9] calculated 38 different software metrics for the open source Apache 1.3 and 2.0 projects. Using PCA, they selected a subset of nine of these metrics which explained 95% of the total data variance.

Also, optimized set reduction (OSR) techniques and logistic regression techniques are used for modeling risk and fault data. OSR [3] attempts to determine which subsets of observations from historical data provide the best characterization of the programs being assessed. Each of these optimal subsets is characterized by a set of predicates (a pattern), which can be applied to classify new programs. OSR is sometimes better than logistic regression analysis for multivariate empirical modeling since pattern-based classification is more accurate than logistic regression equations [3].

Discriminant analysis, a statistical technique used to categorize programs into groups based on the metric values, has been used as a tool for the detection of fault-prone programs [14, 17, 21]. Munson et al. [21] used discriminant analysis for classifying programs as fault-prone within a large medical-imaging software system. In their analysis, they also employed the data-splitting technique, where subsets of programs are selected at random and used to train or build a model. The remaining programs are used to quantify the error in estimation of the number of faults. The data splitting technique is employed to get an independent assessment of how well the reliability can be estimated from a random population sample. The classification resulting from Munson's use of discriminant analysis/data splitting was fairly accurate; there were 10% false positives among the high quality programs (incorrectly classified as fault-prone) and 13% false negatives (incorrectly classified as not fault-prone) among the fault-prone programs. In this paper we use discriminant analysis to classify components as fault-prone or not based on the static analysis tool found defects.

Regression trees and classification trees [19] have also been used to identify fault and not-fault prone modules. Case studies performed using regression trees on a large scale industrial system [13] and classification trees[16] on multiple releases of a large scale legacy telecommunications system indicate the efficacy of regression and classification trees towards identification of fault-prone components. Other such techniques include utilizing neural networks for identifying fault-prone modules for extra attention early in software development [15]. A comprehensive evaluation of fault prediction modeling is discussed by Khoshgoftaar and Seliya. [19]

These studies have used structural and complexity metrics of the source code to act as predictors of system defect density and fault-proneness. Instead, we leverage the subset of defects found by static analysis tools to estimate pre-release system defect density. While static analysis tools find only a subset of all the actual defects, it is highly likely that these defects would be indicative of the overall code quality. Our basic hypothesis is that the more

static analysis defects found, the higher the probability of other defects being present.

A similar approach was used in a study carried out at Nortel Networks on an 800 KLOC commercial software system [23]. Automatic inspection defects found by static analysis tools along with code churn was found to be a statistically significant indicator of field failures and is effective in identifying fault-prone modules.

3. PREfix AND PREfast AT MICROSOFT

We give a brief overview of the PREfix and PREfast tools used within Microsoft. Over 12.5 percent of the defects fixed in Windows Server 2003 before it was released were found with the PREfix and PREfast tools [20]. They represent the state-of-the-art in industrial static analysis tools.

3.1 PREfix

The PREfix tool symbolically executes select paths through a C/C++ program. During this symbolic execution it looks for a multitude of common low-level programming errors, including NULL pointer dereferences, the use of uninitialized memory, double freeing of resources, etc.

PREfix starts with the leaf procedures (such as "foo") in the call graph of a program. It selects a set of paths to cover the statements in procedure foo and symbolically executes these paths. Based upon this analysis, it creates a partial symbolic summary of procedure foo's behavior. It then performs the same sort of analysis the next level up in the call graph. When PREfix encounters a call to procedure foo, it does not reanalyze the body of the procedure foo but uses the symbolic summary in its place. PREfix uses various heuristics to rule out infeasible execution paths.

The PREfix tool has been applied to the Windows code base for the past six years. Its usefulness can be measured by the fact that the errors PREfix detects are automatically entered into a defect database to be fixed by programmers. The PREfix analysis is not inexpensive, requiring a server-side solution for effective deployment on as massive a code base as Windows.

3.2 PREfast

The PREfast tool is a "fast" version of the PREfix tool. Its development was independent of the PREfix tool but aimed at desktop deployment. As a result, the PREfast analyses are inexpensive, accounting for negligible percentage of compile time. Certain PREfast analyses are based on pattern matching in the abstract syntax tree of the C/C++ program to find simple programming mistakes. Other analyses are based on local dataflow analyses to find uninitialized use of variables, NULL pointer dereferences, etc.

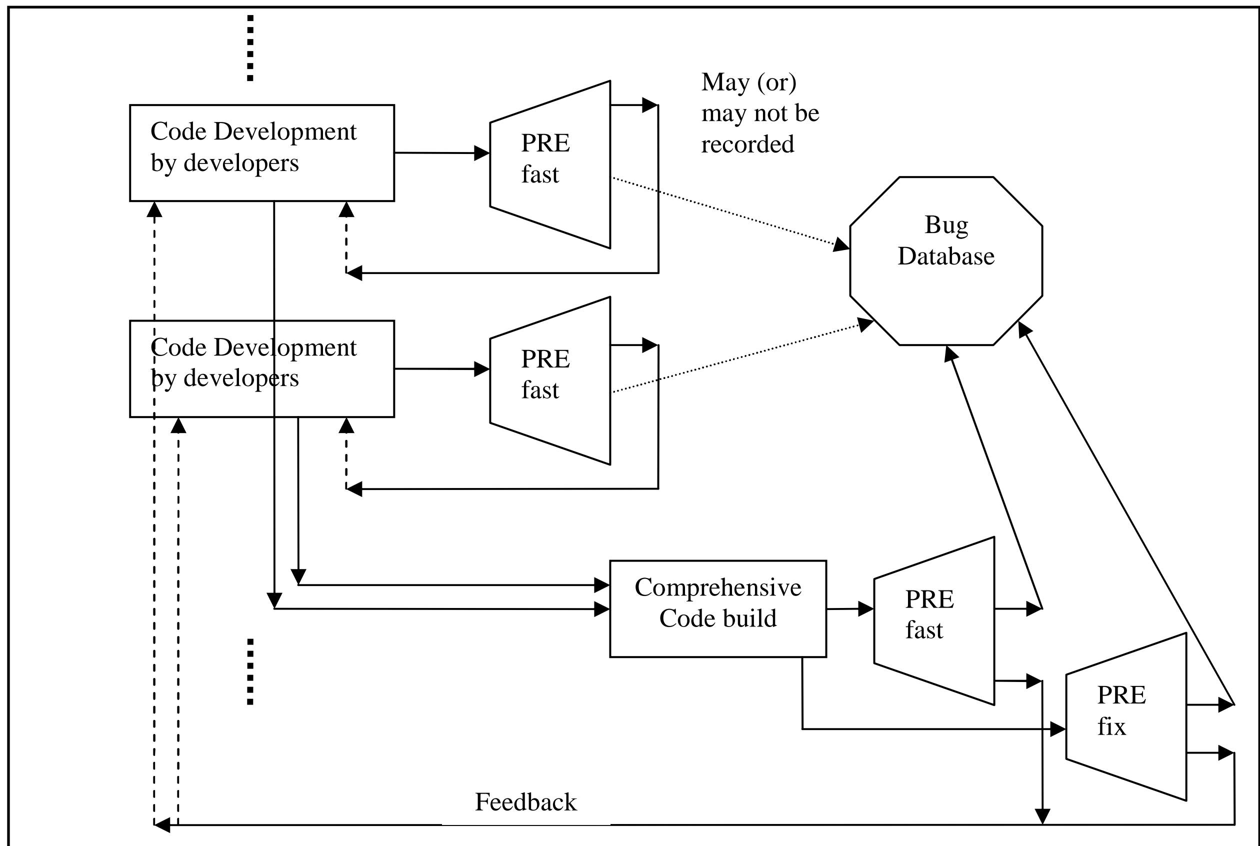

Figure 1: Software Development Process involving PREfast and PREfix

3.3 The Development Process

As mentioned above, PREfast's main use is on the developers' desktops. In this usage scenario, defects found may or may not be recorded by the developer depending on their severity. We expect that many PREfast errors will not be recorded in the defect database. Figure 1 below explains the deployment of PREfast and PREfix in the development process in Microsoft.

At specific points in time all the code from the developers is integrated into a single build that is run through a PREfast run in order to catch all the remaining PREfast errors, as well as a PREfix run. The results of these runs are then automatically entered into a defect tracking database.

4. CASE STUDY

The case study below details the analysis that was performed to evaluate our hypotheses. The experiment was carried out using 199 components of Windows Server 2003. These components had a collective size of 22M LOC (22,000 KLOC). The PREfix and PREfast defects for these components were extracted on a component basis from the defect database. The other "pre-release" defects extracted from the defect database are a comprehensive collection coming from testing teams, integration teams, build results, external teams, third party testers etc. These defects do not include customer defects which are found only in a post-release scenario. The PREfast defects per component, normalized per KLOC, is denoted by "PREfast defect density". We similarly define PREfix and pre-release defect density.

4.1 Correlations

Table 1 shows the correlation results of a Spearman rank correlation between the defect densities of PREfast and PREfix found defects with the pre-release defect densities. Spearman rank correlation is a commonly used robust correlation technique [8] because it can be applied even when the association between elements is non-linear. Table 1 shows a statistically significant[1] positive correlation between the PREfast, PREfix and pre-release defect densities. That is, an increase in the defect densities of PREfast and PREfix found defects is accompanied by an increase in the pre-release defect density. This indicates the promise of using PREfix and PREfast defects as early indicators of pre-release defect density.

[1] SPSS® was used for the purpose of statistical analysis. Through the rest of the paper we assume statistical significance at 99% confidence.

Table 1. Correlation results of Pre-release defects/KLOC (All correlations are significant at the 0.01 level (2-tailed))

	Prefast defects/ KLOC	Prefix defects / KLOC	Pre-release defects/ KLOC
Prefast defects /KLOC (ρ)	1.000		
Prefix defects /KLOC (ρ)	.380	1.000	
Pre-release defects/KLOC (ρ)	**.368**	**.577**	1.000

4.2 Model Building

We use statistical regression techniques to build models to predict the ability of PREfast and PREfix defect densities to estimate the pre-release defect density. We initially fit several models to the PREfast and PREfix data separately as predictors and the pre-release defect density as the dependent variable. The models fit include linear, logarithmic, inverse, quadratic, cubic, power, compound, logistic, growth and exponential models. We then used the PREfast defects/KLOC and the PREfix defects/KLOC together as predictors in a multiple regression model with the pre-release defects as the dependent variable. Table 2 shows the results of the best fits measured by the R^2 value. R^2 is a measure of the fit for the given data set. R^2 measures the variance in the dependant variable that is accounted for by the model built using the predictors [2].

Table 2. Regression Fits

Predictors	**Linear (R^2)**	**Better fits ? (R^2)**
PREfast alone	0.566	Yes. Cubic (0.604)
PREfix alone	0.495	Yes. Cubic (0.514)
Both PREfast and PREfix	0.627	N/A

Table 2 shows that when PREfast defect density is used independently, the best fit is obtained by a cubic equation with R^2= 0.604. The best fit for the PREfix defect density is also obtained using a cubic equation with R^2= 0.514. When PREfast and PREfix are combined together as predictors, the R^2 value increases to 0.627 providing the best fit of the available dataset. Hence, we conclude that using PREfast and PREfix defect densities simultaneously is more beneficial than using them independently to explain pre-release defect density. We do not present the regression equations in order to protect proprietary data.

4.3 Data Splitting

In order to explore the efficacy of using the PREfast and PREfix defect density as predictors of pre-release defect density we use the technique of data splitting [21]. The data splitting technique assesses how well the defect density can be estimated from a random population sample. We used a random sample of 132 components (two-thirds) to build a

multiple regression model and the remaining 67 components to check the predictive ability of the built model. The R^2 for the built equation was 0.806, (F=266.603, p<0.0005). Figure 2 shows the graph of the predicted and actual pre-release defect density. Due to the proprietary nature of the data the axis on the graphs are removed.

In order to quantify the sensitivity of the results we ran a Spearman correlation between the actual and estimated defect densities. The result was a positive correlation of 0.564, (p<0.0005). This indicates that an increase in the predicted value of the PREfast/PREfix defect density is accompanied by an increase in the pre-release defect density at a statistically significant level. The Pearson correlation coefficient is 0.669 (p<0.0005) indicating a similar relationship between the actual and estimated pre-release defect density.

Despite some fluctuations in the observations between the estimated and the actual values, we note that the Y axis is defect density expressed in KLOC, so even a slight change in prediction is reflected by a huge magnitude in the graph.

Analyses that are based on a single dataset that use the same data to both estimate the model and to assess its performance can lead to unreasonably negative biased estimates of sampling variability. In order to address the fact that these results were not by chance we repeated the data splitting experiment. Figure 3 shows three additional predictions of the pre-release defect density using three different random samples. Table 3 shows the results of the Spearman correlation results between the actual and estimated values of pre-release defect density.

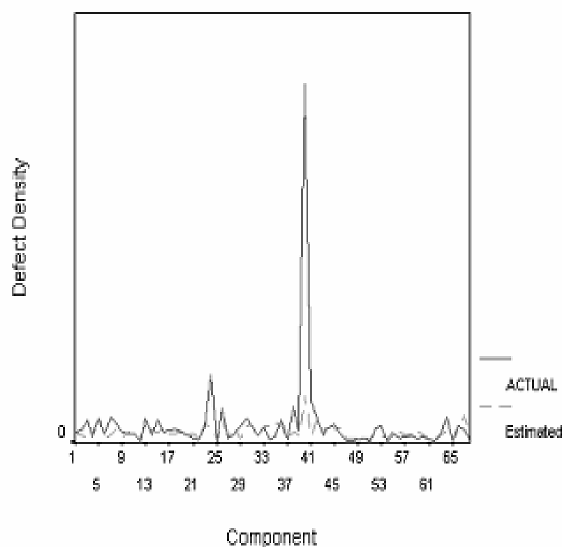

Figure 2. Actual vs. estimated pre-release defect density

Figure 3 and Table 3 confirm that our results were not by chance. The results are similar to Figure 1, where the prediction follows a similar trend as the actual value though there are fluctuations in the prediction.

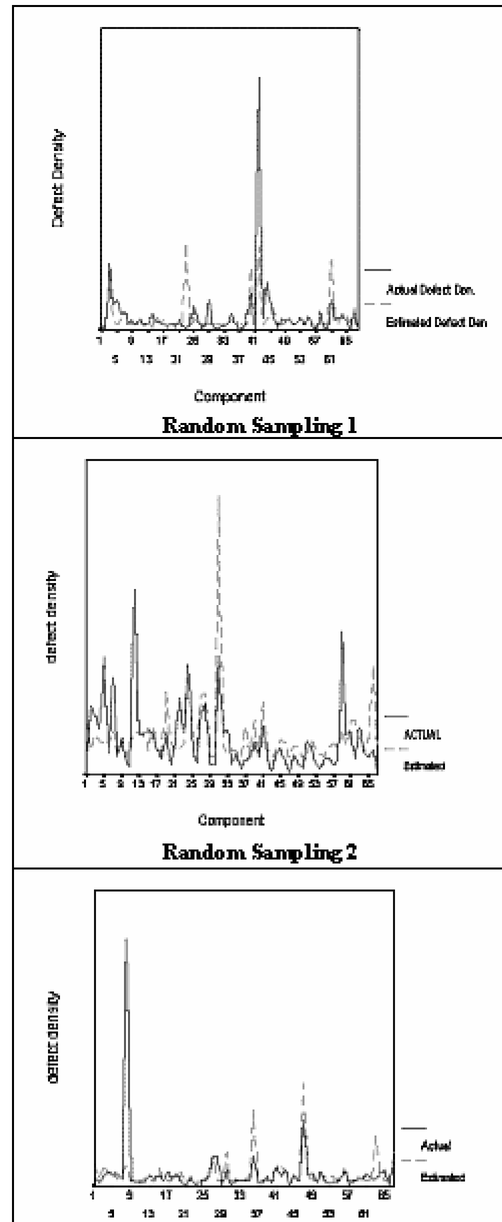

Figure 3. Estimated vs. Actual Defect Pre-Release Defect Density (3 random samples)

Table 3. Fit and Correlation results of random model splitting

S.No	R^2	F-Test (sig)	Correlation Results (Spearman)
1.	0.870	429.79,p<0.0005	0.496, p<0.0005
2.	0.656	339.95,p<0.0005	0.536, p<0.0005
3.	0.841	122.83,p<0.0005	0.526, p<0.0005

4.4 Discriminant Analysis

Discriminant analysis is a statistical technique used to categorize elements into groups based on metric values which has been used to detect fault-prone programs [17, 21]. We use discriminant analysis to identify fault-prone and not fault-prone components using the PREfast and PREfix defect densities as predictors.

Fault-prone and not fault-prone components are calculated from the pre-release defect density of the 199 components using a conservative normal statistical upper cut-off (UCO) formula given in equation 1:

$$UCO = \mu_x + [(z_{\alpha/2} * \text{Standard deviation of pre-release defect density}) / \sqrt{n}]$$

where

- UCO is the upper bound on the pre-release defect density;
- μ_x is the mean of the pre-release defect density;
- $Z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution;
- n is the number of observations.

Based on the above classification, a discriminant function (eigenvalue of 0.135) and a Wilks lambda (0.869) with $\chi^2 = 24.847$, ($p<0.0005$) indicates that the null hypothesis (the value of the discriminant function is the same for both fault-prone and not fault-prone components) can be rejected. These results indicate that the discriminant function built using the predictors adapts itself according to the components.

The overall classification obtained by discriminant analysis is 82.91 % (165 of the 199 components are correctly identified as fault or not fault-prone). The type I and type II misclassifications are not separately reported to protect proprietary information. The overall classification rate indicates the efficacy of identifying fault and not fault-prone components that would lead to appropriate allocation of testing resources, inspections etc.

4.5 Preliminary Validation

Using a metric validation scheme as proposed by Schneidewind [24], we examine our results of using PREfast and PREfix defect density to estimate pre-release defect density. Under Schneidewind's validation scheme, the indicator of quality is the pre-release defect density (F) and the metric suite (M) is comprised of the PREfix and PREfast defect density:

- *Association* tests if there is a sufficient linear association between the F and M to warrant using M as an indirect measure of F. A linear correlation between the pre-release defect density and the PREfast and PREfix defect density results in a statistically significant result demonstrating the association between F and M.
- *Consistency* assesses whether there is sufficient consistency between the ranks of F and M to warrant using M as an indirect measure of F. This is satisfied based on the results obtained in Table 2.
- *Discriminative Power* tests if the metric(s) are able to discriminate between high quality and low quality software components. As we have shown, discriminant analysis effectively separates faulty and not fault-prone components.
- *Tracking* assesses if M is capable of tracking changes in F, i.e. are changes in M reflected by appropriate changes in F. Figure 2, Figure 3 and Table 3 establish the ability of M to track F.
- *Predictability* assesses the predictability of F by M. The correlation between the actual and the estimated results provide evidence for the predictability of F by M.
- *Repeatability* assesses whether the experiment is repeatable. This was demonstrated by using three random splitting samples. One limitation with respect to repeatability is that all the data points are from one software system.

4.6 Limitations of Study

It is possible that PREfix and PREfast might have missed defects that should have been caught during the development process. These defects would be found by testing, thus increasing the pre-release defect density which might skew the correlation. Our results are specific to the PREfix and PREfast tools. The PREfix tool's usefulness can be measured by the fact that the errors PREfix detects are automatically entered into a defect database to be fixed by programmers. Programmers might find some of the entered defects to be false positives. This is alleviated to some extent by the fact that in the absence of manual intervention ("programmers") the static analysis found defects are indicative of pre-release defect density. Further, these results are heavily dependent on the quality of the static analysis tool and might not be repeatable with the same degree of strength for other tools.

5. LESSONS LEARNED AND FUTURE WORK

Based on our results, we find that:

- Static analysis defect density can be used as early indicators of pre-release defect density;
- Static analysis defect density can be used to predict pre-release defect density at statistically significant levels;
- Static analysis defect density can be used to discriminate between components of high and low quality.

The above results allow us to identify areas that are likely to have high pre-release defects that can help inform decisions on testing, code inspections, design rework etc. We further

observe that using both PREfast and PREfix found defects as predictors of pre-release defect density is better than using them separately.

We plan to further validate the system by deploying it so that developers can immediately get feedback on their pre-release defect density. We also plan to include standards for the maximum allowable defects that can be found by these tools for enforcing acceptable code during check-ins. We also are building these models across several software systems in Microsoft to make the prediction models more robust.

ACKNOWLEDGEMENTS

We would like to thank Madan Musuvathi of the Testing, Verification and Measurement group for several discussions and reviews of initial drafts of this paper. We acknowledge, Pankaj Jalote, Visiting Researcher at Microsoft Research from IIT Kanpur, Rajesh Munshi and Naveen Sethuraman from the Windows Team for providing access to their datasets. We would like to thank Jim Larus, Jon Pincus and Manuvir Das of Microsoft Research for their discussions and feedback on PREfix and PREfast, and Bojan Cukic of West Virginia University and the anonymous referees for their thoughtful comments on earlier drafts of this paper.

REFERENCES

[1] Basili, V., Briand, L., Melo, W., "A Validation of Object Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. Vol. 22, pp. 751 - 761, 1996.

[2] Brace, N., Kemp, R., Snelgar, R., *SPSS for Psychologists*: Palgrave Macmillan, 2003.

[3] Briand, L. C., Thomas, W.M., Hetmanski, C.J., "Modeling and managing risk early in software development," Proceedings of International Conference on Software Engineering 2003, 1993.pp 55-65.

[4] Briand, L. C., Wuest, J., Daly, J.W., Porter, D.V., "Exploring the Relationship between Design Measures and Software Quality in Object Oriented Systems," *Journal of Systems and Software*, vol. Vol. 51, pp. 245-273, 2000.

[5] Briand, L. C., Wuest, J., Ikonomovski, S., Lounis, H., "Investigating quality factors in object-oriented designs: an industrial case study," Proceedings of ICSE, 1999.pp 345-354.

[6] Bush, W. R., Pincus, J.D., Sielaff, D.J., "A Static Analyzer for Finding Dynamic Programming Errors," *Software-Practice and Experience*, vol. 20, pp. 775-802, 2000.

[7] Chidamber, S. R. and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, 1994.

[8] Denaro, G., Morasca, S., Pezze, M., "Deriving Models of Software Fault-Proneness," Proceedings of SEKE 2002, 2002.pp 361-368.

[9] Denaro, G., Pezze, M., "An empirical evaluation of fault-proneness models," Proceedings of International Conference on Software Engineering, 2002.pp 241 - 251.

[10] Engler, D., Chelf, B., Chou, A., Hallem, S., "Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions," Proceedings of OSDI 2000, 2000.pp

[11] Evans, D., Guttag, J., Horning, J., Tan, Y., M.,, "LCLint: A Tool for Using Cpecifications to Check Code," Proceedings of ACM-SIGSOFT Foundations in Software Engineering, 1994.pp 87-96.

[12] Fenton, N. E., Pfleeger, S.L., *Software Metrics*. Boston, MA: International Thompson Publishing, 1997.

[13] Khoshgoftaar, T. M., Allen, E.B., Deng,J., "Using Regression Trees to Classify Fault-Prone Software Modules," *IEEE Transactions on Reliability*, vol. 51, pp. 455-462, 2002.

[14] Khoshgoftaar, T. M., Allen, E.B., Goel, N., Nandi, A., McMullan, J., "Detection of Software Modules with high Debug Code Churn in a very large Legacy System," Proceedings of International Symposium on Software Reliability Engineering, 1996.pp 364-371.

[15] Khoshgoftaar, T. M., Allen, E.B., Hudepohl, J.P., Aud, S.J., "Application of neural networks to software quality modeling of a very large telecommunications system," *IEEE Transactions on Neural Networks*, vol. 8, pp. 902-909, 1997.

[16] Khoshgoftaar, T. M., Allen, E.B., Jones,W.D., Hudepohl, J.P., "Classification-Tree Models of Software Quality Over Multiple Releases," *IEEE Transactions on Reliability*, vol. 49, pp. 4-11, 2000.

[17] Khoshgoftaar, T. M., Allen, E.B., Kalaichelvan, K.S., Goel, N., Hudepohl, J.P., Mayrand, J., "Detection of fault-prone program modules in a very large telecommunications system," Proceedings of International Symposium Software Reliability Engineering, 1995.pp 24-33.

[18] Khoshgoftaar, T. M., Munson, J.C., Lanning, D.L., "A Comparative Study of Predictive Models for Program Changes During System Testing and Maintenance," Proceedings of, International Conference on Software Maintenance, 1993.pp 72-79.

[19] Khoshgoftaar, T. M., Seliya, N., "Fault Prediction Modeling for Software Quality Estimation : Comparing Commonly Used Techniques," *Empirical Software Engineering*, vol. 8, pp. 255-283, 2003.

[20] Larus, J. R., Ball, T., Das, M., DeLine, R., Fahndrich, M., Pincus, J., Rajamani, S.K., Venkatapathy, R., "Righting Software," in *IEEE Software*, vol. 21, 2004, pp. 92-100.

[21] Munson, J. C., Khoshgoftaar, T.M., "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering*, vol. 18, pp. 423-433, 1992.

[22] Munson, J. C., Khoshgoftaar,T.M.,, "Regression Modeling of Software quality : Empirical Investigation," *Information and Software Technology*, pp. 106-114, 1990.

[23] Nagappan, N., Williams, L., Hudepohl, J., Snipes, W., Vouk, M., "Preliminary Results On Using Static Analysis Tools For Software Inspection," Proceedings of Fifteenth IEEE International Symposium on

Software Reliability Engineering, St. Malo, France, 2004.pp 429-439.
[24] Schneidewind, N. F., "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering*, vol. 18, pp. 410-422, 1992.
[25] Subramanyam, R., Krishnan, M.S., "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Transactions on Software Engineering*, vol. Vol. 29, pp. 297 - 310, 2003.
[26] Tang, M.-H., Kao, M-H., Chen, M-H., "An empirical study on object-oriented metrics," Proceedings of Sixth International Software Metrics Symposium, 1999.pp 242-249.