

Early Estimation of Software Quality Using In-Process Testing Metrics: A Controlled Case Study

Nachiappan Nagappan¹, Laurie Williams², Mladen Vouk², Jason Osborne³

¹ Microsoft Research, Redmond, WA 98052
nachin@microsoft.com

² Department of Computer Science, North Carolina State University, Raleigh, NC 27695

³ Department of Statistics, North Carolina State University, Raleigh, NC 27695
{lawilli3, vouk, jaosborn}@ncsu.edu

ABSTRACT

In industrial practice, information on post-release field quality of a product tends to become available too late in the software development process to affordably guide corrective actions. An important step towards remediation of this problem of late information lies in the ability to provide an early estimation of software post-release field quality. This paper presents the use of a suite of in-process metrics that leverages the software testing effort to provide (1) an estimation of potential software field quality in early software development phases, and (2) the identification of low quality software programs. A controlled case study conducted at North Carolina State University provides initial indication that our approach is effective for making an early assessment of post-release field quality.

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Metrics - Performance measures, Process metrics, Product metrics.

General Terms

Measurement, Design, Reliability.

Keywords

Testing metrics, empirical software engineering, multiple regression, software field quality.

1. INTRODUCTION

In industry, estimates of software field quality are often available too late to affordably guide corrective actions to the quality of the software. True field quality cannot be measured before a product has been completed and delivered to an internal or external customer. Field quality is calculated using the number of failures found by these customers. Because this information is available

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

3-WoSQ '05, May 17, 2005, St Louis, Missouri, USA.

Copyright 2005 ACM 1-59593-122-8/05/0005...\$5.00.

late in the process, corrective actions tend to be expensive. [3] Software developers can benefit from an early warning regarding the quality of their product.

This early warning can be built from a collection of internal metrics that are correlated with field quality, an external measure. An internal metric, such as the cyclomatic complexity [20], is a measure derived from the product itself [13]. An external measure is a measure of a product derived from the external assessment of the behavior of the system [13]. For example, the number of defects found in test is an external measure.

Structural object-orientated (O-O) measurements, such as those defined in the CK [9] and MOOD [7] O-O metric suites, are being used to evaluate and predict the quality of software [12]. Structural O-O measurements, such as those in the Chidamber-Kemerer (C-K) O-O metric suite [9], have been used to evaluate and predict fault-proneness [1, 5, 6]. The CK metric suite consists of six metrics: weighted methods per class (WMC), coupling between objects (CBO), depth of inheritance tree (DIT), number of children (NOC), response for a class (RFC) and lack of cohesion among methods (LCOM). These metrics can be a useful early internal indicator of externally-visible product quality in terms of fault-proneness [1, 27, 28].

To summarize, there is a growing body of empirical evidence that supports the theoretical validity of the use of these internal metrics [1, 5] as predictors of fault-proneness. The consistency of these findings varies with the programming language [27]. Therefore, the metrics are still open to criticism. [10]

Our research objective is to construct and validate a set of easy-to-measure in-process metrics that can be used as an early indication of an external measure of field quality. To this end, we have created a metric suite we call the Software Testing and Reliability Early Warning metric suite for Java (STREW-J)[22]. The metric suite is applicable for development teams that write extensive automated test cases, such as is done in the Extreme Programming [2] software development methodology. In this paper, we present the results of a study designed to analyze the capabilities of the STREW metrics suite. The study was run with junior- and senior-students in a software engineering class at North Carolina State University (NCSU).

The paper is organized as follows. Section 2 outlines the STREW metric suite. Sections 3 discusses a controlled experiment, in which we studied the STREW metric suite. Section 4 presents the

experimental results. Finally, Section 5 presents the conclusions and future work.

2. STREW METRIC SUITE

The STREW-J metric suite is a set of internal, in-process software metrics that are leveraged to make an early estimation of post-release field quality. The reasoning behind the metric suites is different from the traditional reliability estimation [31] models is that STREW puts a greater emphasis on internal software metrics, particularly those involving the testing effort. The metrics are intended to cross-check each other and to triangulate upon a post-release field quality estimate. Prior studies [1, 5, 6, 8, 11, 27, 28, 30] have all leveraged the structural aspects of the code, but not the testing effort that has been carried out, to make an estimate of field quality.

The use of the STREW metrics is predicated on the existence of an extensive suite of automated unit test cases being created as development proceeds. STREW leverages the utility of automated test suites by providing a post-release field quality estimate. During the initial stages of creating a system, however, such a test suite might not be available. In that case, historical data from a comparable project may be used. The field quality estimate relative to historical data is calculated using multiple linear regression analysis which is used to model the relationship between software quality and selected software metrics [17, 21].

The STREW-J metric suite consists of nine constituent metric ratios. The metrics are intended to cross-check each other and to triangulate upon an estimate of post-release field quality. Each metric makes an individual contribution towards estimation of the post-release field quality but work best when used together. Development teams record the values of these nine metrics and the actual post-release field quality of projects. These historical values from prior projects are used to build a regression model that is used to estimate the post-release field quality of the current project under development. The nine constituent STREW-J metrics (SM1 – SM9) are shown below in Table 1. The metrics can be categorized into three groups: test quantification metrics, complexity and O-O metrics, and a size adjustment metric.

Table 1: STREW metrics and collection and computation instructions

Test quantification	
$\frac{\text{Number of Assertions}}{SLOC^*}$	SM1
$\frac{\text{Number of Test Cases}}{SLOC^*}$	SM2
$\frac{\text{Number of Assertions}}{\text{Number of Test Cases}}$	SM3
$\frac{(TLOC^+ / SLOC^*)}{(\text{Number of Classes}_{Test} \text{ Number of Classes}_{Source})}$	SM4
Complexity and O-O metrics	
$\frac{\sum Cyclomatic Complexity_{Test}}{\sum Cyclomatic Complexity_{Source}}$	SM5
$\frac{\sum CBO_{Test}}{\sum CBO_{Source}}$	SM6
$\frac{\sum DIT_{Test}}{\sum DIT_{Source}}$	SM7
$\sum WMC_{Test}$	SM8

$\sum WMC_{Source}$	
Size adjustment	
$\frac{SLOC^*}{\text{Minimum } SLOC^*}$	SM9
* Source Lines of Code (SLOC) is computed as non-blank, non-comment source lines of code	
+ Test Lines of Code (TLOC) is computed as non-blank, non-comment test lines of code	

The **test quantification metrics** (SM1, SM2, SM3, and SM4) are specifically intended to crosscheck each other to account for coding/testing styles. For example, one developer might write fewer test cases, each with multiple asserts [26] checking various conditions. Another developer might test the same conditions by writing many more test cases, each with only one assert. We intend for our metric suite to provide useful guidance to each of these developers without prescribing the style of writing the test cases. Assertions [26] are used in two of the metrics as a means for demonstrating that the program is behaving as expected and as an indication of how thoroughly the source classes have been tested on a per class level. SM4 serves as a control measure to counter the confounding effect of class size (as shown by El-Emam [11]) on the prediction efficiency;

The **complexity and O-O metrics** (SM5, SM6, SM7, and SM8) examines the relative ratio of test to source code for control flow complexity and for a subset of the CK metrics. The dual hierarchy of the test and source code allows us to collect and relate these metrics for both test and source code. These relative ratios for a product under development can be compared with the historical values for prior comparable projects to indicate the relative complexity of the testing effort with respect to the source code. The metrics are now discussed more fully:

- The *cyclomatic complexity* [20] metric for software systems is adapted from the classical graph theoretical cyclomatic number and can be defined as the number of linearly independent paths in a program. Prior studies have found a strong correlation between the cyclomatic complexity measure and the number of test defects [29]. Studies have also shown that code complexity correlates strongly with program size measured by lines of code [15] and is an indication of the extent to which control flow is used. The use of conditional statements increases the amount of testing required because there are more logic and data flow paths to be verified [16].
- The larger the inter-object *coupling*, the higher the sensitivity to change [9]. Therefore, maintenance of the code is more difficult [9]. Prior studies have shown CBO has been shown to be related to fault-proneness [1, 5, 6]. As a result, the higher the inter-object class coupling, the more rigorous the testing should be [9].
- A higher *DIT* indicates desirable reuse but adds to the complexity of the code because a change or a failure in a super class propagates down the inheritance tree. The relationship between the DIT and fault-proneness [1, 5] was found to be strongly correlated.
- The *number of methods* and the *complexity of methods* involved is a predictor of how much time and effort is required to develop and maintain the class [9]. The larger the number of methods in a class, the greater is the potential impact on children, since the children will

inherit all the methods defined in the class. The ratio of the WMC_{test} and WMC_{source} measures the relative ratio of the number of test methods to source methods. This measure serves to compare the testing effort on a method basis. The relationship between the WMC as an indicator of fault-proneness has been demonstrated in prior studies[1, 5].

The final metric is a **relative size adjustment factor**. Defect density has been shown to increase with class size [11]. We account for the difference in size in terms of lines of code for the projects used to build the STREW prediction equation using the size adjustment factor.

The metrics that comprise the STREW metric suite have evolved [23-25] through our case studies. Some metrics were removed based on the lack of their ability to contribute towards the estimation of post-release field quality and due to already existing inter-correlations between the elements. The removal of metrics was governed by statistical inter-correlations, multicollinearity, stepwise, backward, and forward regression techniques [19]. The following metrics were removed:

- Statement coverage
- Branch coverage
- Number of requirements/Source lines of code
- Number of children_{test}/Number of children_{source}
- Lack of cohesion among methods_{test}/Lack of cohesion among methods_{source}

3. CONTROLLED EXPERIMENT

Section 3.1 discusses the research design of the controlled experiment performed in an academic environment at North Carolina State University (NCSU). The experimental limitations are discussed in section 3.2.

3.1. Research Design

To evaluate the predictive ability of STREW-J to provide an early estimate of post-release field quality, a case study was carried out in a junior/senior-level software engineering course at NCSU in the fall 2003 semester. Students developed an open source Eclipse¹ plug-in in Java that automated the collection of static code metrics. Each project was developed by a group of four or five junior or senior undergraduates during a six-week final class project. The plug-in was required to have 80% unit test coverage via the JUnit² unit test suite and for acceptance test cases to be automated via the FIT³ tool. A total of 22 projects were submitted; all were used in the analysis. Table 2 below shows the size of the projects developed in terms of the source lines of code (SLOC) and the test lines of code (TLOC).

Table 2: Project size

Metric	Mean	Std Dev	Max	Min
SLOC	1996.9	835.9	3631	617
TLOC	688.7	464.4	2115	156

¹ Eclipse is an open source integrated development environment. For more information, go to <http://www.eclipse.org/>.

² <http://junit.org/index.htm>

³ <http://fit.c2.com/>

The plug-ins were evaluated using a comprehensive set of 45 black-box test cases. Twenty-six of these were acceptance tests and were given to the students during development. The 45 test cases included exception checking, error handling, and boundary test cases that checked the operational correctness of the plug-in.

3.2. Case Study Limitations

An external validity issue arises because the programs were written by students. This concern is mitigated to some degree because the students were advanced undergraduates (junior/senior). Some of the students have had industry experience. Since the experiment was performed in an academic setting it was possible to exert a degree of control that could ensure uniformity across the development environment. This control may represent ideal conditions to study the involved dependent and independent variables that might not be exactly reflective of industrial software development.

The Eclipse plug-ins are small relative to industry applications. Additionally, we calculated actual post-release field quality by running black box test cases and computing an approximation of field quality (black box test failures/KLOC). Black box test failures are approximated as the problems that would have been found by the customer had the academic projects been released to an external customer. Further, using the same set of 45 test cases to evaluate post-release field quality may also have skewed the Black box test failures/KLOC to a certain degree as all the groups might not have made the same kind of failures.

4. EXPERIMENTAL RESULTS

Using the post-release field quality calculated via black box test failures/KLOC obtained by running the 45 test cases, as the dependant variable and the seven metrics as predictors, a multiple linear regression analysis on was performed. One difficulty associated with multiple linear regression is multicollinearity among the metrics, which can lead to inflated variance in the prediction of post-release field quality. Multicollinearity occurs because of internal correlations between the metrics as shown in Table 4. To address the potential problem caused by multicollinearity, we perform PCA [14] using the above 22 projects. In PCA, a smaller number of uncorrelated linear combination of the metrics (principal components) are created for use in a regression analysis. These principal components do not suffer from multicollinearity and accounts for as much sample variance as possible.

Table 3: Principal Component Matrix

	Component		
	1	2	3
SM1	.509	.590	.380
SM2	.438	.544	-.536
SM3	.272	-3.501E-02	.877
SM4	.365	.808	-.176
SM5	.784	-.560	-9.552E-02
SM6	.727	-.482	-9.762E-02
SM7	.661	.513	-2.921E-03

SM8	.645	-.573	3.544E-02
SM9	-1.454E-02	.278	.584

Performing PCA resulted in the reduction of the seven metric measures into three principal components. These components

remove multicollinearity and are orthogonal to each other. This means that changes in one component do not influence either of the other components, unlike the individual metrics.

Table 4: Correlation Matrix with Black box test failures/KLOC

		SM1	SM2	SM3	SM4	SM5	SM6	SM7	SM8	SM9	Black box test failures/KLOC
SM1	r	1									
	Sig.	.									
SM2	r	.450	1								
	Sig.	.036	.								
SM3	r	.569	-.310	1							
	Sig.	.006	.160	.							
SM4	r	.510	.564	-.095	1						
	Sig.	.015	.006	.675	.						
SM5	r	.011	.096	.097	-.127	1					
	Sig.	.963	.670	.668	.573	.					
SM6	r	.021	.069	.153	-.127	.819	1				
	Sig.	.927	.760	.497	.572	.000	.				
SM7	r	.447	.402	.045	.660	.205	.283	1			
	Sig.	.037	.063	.842	.001	.361	.202	.			
SM8	r	-.020	-.032	.183	-.184	.834	.520	.116	1		
	Sig.	.929	.886	.416	.412	.000	.013	.606	.		
SM9	r	.106	-.169	.239	.121	-.112	-.214	.281	-.087	1	
	Sig.	.637	.452	.285	.590	.619	.340	.205	.700	.	
Black box test failures/KLOC	r	-.185	.226	-.307	-.103	.393	.642	.101	.118	-.627	1
	Sig.	.410	.312	.165	.647	.071	.001	.655	.600	.002	.

In order to demonstrate the efficacy of the predictions, of the black box test failures/KLOC, we use the regression predicted values of the three principal components (PC1, PC2, and PC3) from all the 22 projects to build a regression model, as shown in Equation 1.

$$\text{Black box test failures/KLOC} = 7.60 + 2.36*PC1 - 2.40*PC2 - 4.39*PC3 \dots (1)$$

The F-ratio to test the hypothesis that all regression coefficients of the model are zero was statistically significant, ($R^2 = 0.512$, ($F=6.284$, $p=0.004$)). The multiple coefficient of determination, R^2 , provides a quantification of how much variability in the quality can be explained by the regression model. But this R^2 does not quantify the predictability of the STREW-J metric suite. For this, we employ a technique of random data splitting. From the available 22 projects, 15 projects were randomly selected to build a different multiple regression model, and the model was used to predict the defect density of the remaining seven projects to

evaluate the fit of the model in terms of predictability. We repeat the random split seven times to ensure that our results are not for one particular case. The predictability of the models is computed using the average absolute error (AAE)[18] and average relative error (ARE)[18] and shown in Equation 2 and 3.

$$AAE = \frac{1}{n} \sum_{i=1}^n | \text{Estimated Value} - \text{Actual Value} | \dots (2)$$

$$ARE = \frac{1}{n} \sum_{i=1}^n (| \text{Estimated Value} - \text{Actual Value} |) / \text{Actual Value} \dots (3)$$

The AAE and ARE results in Table 5 indicate that the model produces an estimate that is indicative of the true black box test failures/KLOC.

Table 5: AAE, ARE values for random splits

Random Sample	Model characteristics R ² (F- ratio, sig.)	AAE	ARE
1.	0.513 (F=3.867, p=0.041)	2.90	1.16
2.	0.509 (F=3.798, p=0.043)	2.04	0.97
3.	0.445 (F=2.946, p=0.080)	7.49	0.82
4.	0.788 (F=13.621, p=0.001)	7.56	2.10
5.	0.853 (F=21.316, p<0.0005)	6.37	1.68
6.	0.633 (F=6.312, p=0.010)	5.33	1.20
7.	0.690 (F=8.157, p=0.004)	4.50	0.87

The actual values of black box test failures/KLOC range from 1.43 black box test failures/KLOC to 35.5 black box test failures/KLOC indicating a wide spectrum of quality in terms of black box test failures for the academic projects. This range of Black box test failures/KLOC of the projects is shown by the histogram in Figure 1.

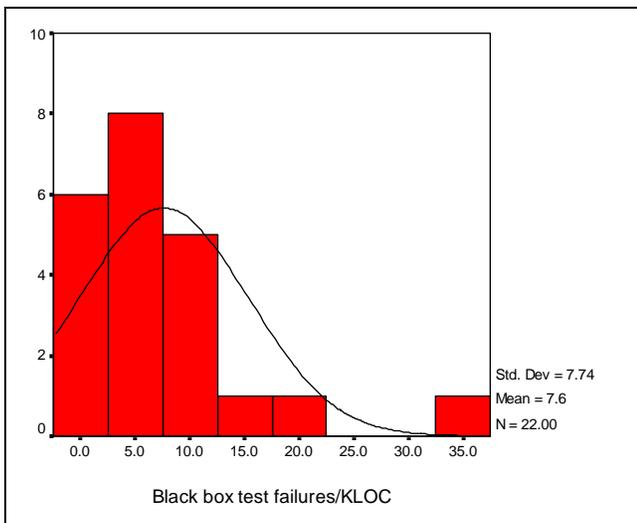


Figure 1: Black box test failures/KLOC

The ARE and AAE values are less than the standard deviation of the black box test failures/KLOC (standard deviation is 7.738 black box test failures/KLOC). In spite of the wide spectrum of the Black box test failures/KLOC, the ARE and AAE values indicate the feasibility of using the STREW metric suite compared to the actual values of the Black box test failures/KLOC. Figure 2 demonstrates that the estimated post-release field quality is a practical estimate of the actual post-release field quality. This indicates the efficacy of using the STREW-J metrics to make an early assessment of post-release field quality.

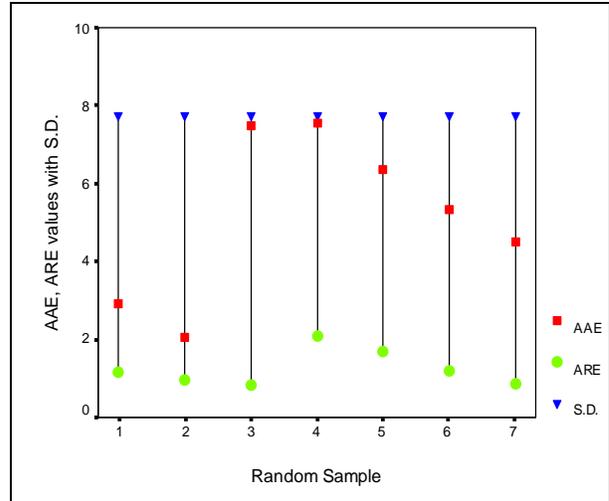


Figure 2: Plots of AAE, ARE with Standard Deviation

Further, to assess the ability of STREW-J metrics to be used to identify programs of low quality, we use the binary logistic regression technique[4]. For logistic regression, the exact R² values cannot be calculated. An approximate measure of R² given by the Cox and Snell R² (0.586) and the Nagelkerke R² (0.790)[4] is used. To determine the efficacy of identifying high and low quality components correctly, we use the statistical normal lower confidence bound formula on the black box test failures given by Equation 4.

$$(\text{lower bound}) = \mu_{\text{Black box test failures}} - [(Z_{\alpha/2} * \text{Standard deviation of black box test failures/KLOC}) / \sqrt{n}] \dots (4)$$

where $Z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution and n is the number of samples.

All programs having a Black box test failures/KLOC lower than the calculated lower bound from Equation 4 are of high quality and the remaining is of low quality. The overall classification of the low and high quality programs in shown in Table 6.

Table 6: Overall classification of program quality

		Predicted Quality		Percentage Correct
Observed Quality		High quality	Low quality	
	High quality	7	2	77.8 CI=[49.05,100]
	Low quality	0	13	100.0
Overall Percentage				90.9 CI= [78.30,100]

The point estimate of the percentage correct classification is 90.9% (i.e. overall 20 of the 22 programs were correctly identified as high or low quality programs.). The confidence intervals (CI) for these point estimates are given in brackets.

5. CONCLUSIONS AND FUTURE WORK

Feedback on potential field quality of software is very useful to developers because it helps identify weaknesses and faults in the software that require fixing. However, in most production environments, field quality is measured too late to affordably guide significant corrective actions. In this paper we have reported on the use of an in-process testing metric suite for providing an early warning regarding post-release field quality measured by black box test failures/KLOC, and for identifying low quality programs.

The main observations are:

- A empirical multiple regression approach using the STREW metric suite is a practical approach to measuring software post-release field quality ; and
- STREW-based logistic regression analysis is a feasible technique for detecting low quality programs.

We will continue to validate the metric suite under different industrial and academic environments. There also arises the need to generalize the STREW-J for other object-oriented languages with more case studies. Finally, we will develop standards for these metrics, so that developers can compare their program quality in a meaningful way, and get feedback on the quality of their testing effort relative to industry developed standards.

ACKNOWLEDGEMENTS

We would like to thank Lucas Layman and Jiang Zheng of NCSU for reviewing earlier drafts of this paper. This work was funded by an in part by an IBM Eclipse Innovation Award and by the National Science Foundation under CAREER Grant No. 0346903. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

- [1] Basili, V., Briand, L., Melo, W., "A Validation of Object Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, pp. 751 - 761, 1996.
- [2] Beck, K., *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.
- [3] Boehm, B. W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- [4] Brace, N., Kemp, R., Snelgar, R., *SPSS for Psychologists*: Palgrave Macmillan, 2003.
- [5] 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. 51, No. 3, pp. 245-273, 2000.
- [6] Briand, L. C., Wuest, J., Ikonomovski, S., Lounis, H., "Investigating Quality Factors in Object-Oriented Designs : An Industrial Case Study," Proceedings of International Conference on Software Engineering, 1999, pp. 345-354.
- [7] Brito e Abreu, F., "The MOOD Metrics Set," Proceedings of ECOOP '95 Workshop on Metrics, 1995, pp.
- [8] Chidamber, S. R., Darcy, D.P., Kemerer, C.F., "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis," *IEEE Transactions on Software Engineering*, Vol. 24, No. 8, pp. 629-639, 1998.
- [9] Chidamber, S. R., Kemerer, C.F., "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476 - 493, 1994.
- [10] Churcher, N. I. and M. J. Shepperd, "Comments on 'A Metrics Suite for Object-Oriented Design'," *IEEE Transactions on Software Engineering*, Vol. 21, No. 3, pp. 263-5, 1995.
- [11] El Emam, K., Benlarbi, S., Goel, N., Rai, S.N., "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering*, Vol. 27, No. 7, pp. 630 - 650, 2001.
- [12] Harrison, R., S. J. Counsell, and R. V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Transactions on Software Engineering*, Vol. 24, No. 6, pp. 491-496, June 1998.
- [13] ISO/IEC, "DIS 14598-1 Information Technology - Software Product Evaluation," 1996.
- [14] Jackson, E. J., *A User's Guide to Principal Components*: John Wiley & Sons, Inc., 1991.
- [15] Kan, S. H., *Metrics and Models in Software Quality Engineering*. Reading, MA: Addison-Wesley, 1995.
- [16] Khoshgoftaar, T. M., Munson, J.C., "Predicting software development errors using software complexity metrics," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 2, pp. 253-261, 1990.
- [17] 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.
- [18] Khoshgoftaar, T. M., Seliya, N., "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques," *Empirical Software Engineering*, Vol. 8, No. 3, pp. 255-283, 2003.
- [19] Kleinbaum, D. G., Kupper, L.L., Muller, K.E., *Applied Regression Analysis and Other Multivariable Methods*. Boston: PWS-KENT Publishing Company, 1987.
- [20] McCabe, T. J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, pp. 308-320, 1976.
- [21] Munson, J. C., Khoshgoftaar, T.M., "Regression Modeling of Software quality : Empirical Investigation," *Information and Software Technology*, Vol. 32, No. 2, pp. 106-114, 1990.
- [22] Nagappan, N., "A Software Testing and Reliability Early Warning (STREW) Metric Suite", Department of Computer Science, North Carolina State University, 2005.
- [23] Nagappan, N., Williams, L., Vouk M.A., ""Good Enough" Software Reliability Estimation Plug-in for Eclipse," Proceedings of IBM-ETX Workshop, in conjunction with OOPSLA 2003, 2003, pp. 36-40.
- [24] Nagappan, N., Williams, L., Vouk M.A., "Towards a Metric Suite for Early Software Reliability Assessment," Proceedings of International Symposium on Software Reliability Engineering, FastAbstract, Denver, CO, 2003, pp. 238-239.
- [25] Nagappan, N., Williams, L., Vouk M.A., Osborne, J., "Using In-Process Testing Metrics to Estimate Software Reliability: A Feasibility Study," Proceedings of IEEE International Symposium on Software Reliability Engineering, FastAbstract, Saint Malo, France, 2004, pp. 21-22.

- [26] Rosenblum, D. S., "A practical approach to programming with assertions," *IEEE Transactions on Software Engineering*, Vol. 21, No. 1, pp. 19-31, 1995.
- [27] 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. 29, No. 4, pp. 297 - 310, 2003.
- [28] 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.
- [29] Troster, J., "Assessing Design-Quality Metrics on Legacy Software," Software Engineering Process Group, IBM Canada Ltd. Laboratory, North York, Ontario 1992.
- [30] Vouk, M. A., Tai, K.C., "Multi-Phase Coverage- and Risk-Based Software Reliability Modeling," Proceedings of CASCON '93, 1993, pp. 513-523.
- [31] Xie, M., *Software Reliability Modeling*. Singapore: World Scientific Publishing Co. Pte. Ltd., 1991.