Journal of International Technology and Information Management

Volume 16 | Issue 3 Article 6

2007

Discovering Software Reliability Patterns Based On Multiple Software Projects

Yi Liu
Georgia College and State University

Gerald Adkins
Georgia College & State University

Jeng-Foung Yao
Georgia College & State University

Gita Williams
Georgia College and State University

Discovering Software Reliability Patterns Based On Multiple Software Projects

Yi Liu
Gerald Adkins
Jeng-Foung Yao
Gita Williams
Georgia College and State University

ABSTRACT

Discovering patterns that indicate software reliability provides valuable information to software project managers. Software Quality Classification (SQC) modeling is a methodology that can be used to discover reliability patterns of large software projects. However, the patterns found by SQC modeling may not be accurate and robust owing to insufficient information used in the training process. This study compares two genetic programming-based SQC models using different volumes of data. These data were extracted from seven different NASA software projects. The results demonstrate that combining data from different projects can produce more accurate and reliable patterns.

INTRODUCTION

As one of the major measurements of software quality (Kitchenham, 1996), software reliability plays a key role in the success of safety-critical systems. Until now, none of the large software systems can be recognized as defect-free regardless of the amount of time and effort invested in the debugging and testing phases of the software development process. Therefore, a critical issue for project managers is to define a reliability goal and achieve the goal within a given time and budget.

Software Quality Classification (SQC) modeling (Khoshgoftaar, Seliya, & Herzberg, 2005) is to discover reliability patterns which categorize software modules in a project as high-risk or low-risk based on software metrics. By using these metrics, SQC modeling can predict the reliability of each software module in early stages of development. Since the effort to locate and correct faults early is much more cost-effective than in later stages, SQC modeling can help project managers to achieve their reliability goals within the given time and cost. Several studies indicate that using this methodology may lead to good results (Nagappan, Ball, & Murphy, 2006; Fenton & Ohlsson, 2000; Khoshgoftaar & Allen, 2000; Briand, Melo, & Wuest, 2002). For example, Nagappan, Ball and Murphy (Nagappan, Ball, & Murphy, 2006) applied the method to Microsoft XP and Windows Server 2003, and concluded the efficacy of building such models to estimate post-release failures and failure proneness. Tian and Palma (Tian & Palma, 1998) introduced a software tool built by using SQC modeling and applied the tool on five large software systems developed by IBM.

Software projects are usually implemented by different teams, in different organizations, using multiple programming languages and methods. Data extracted from different projects may not be consistent, relevant, or follow the same pattern. Therefore, existing approaches to build SQC models always use a single dataset extracted from a single software project. However, the available information based on a single project is very limited, despite the fact that large volumes of information crucial in data mining may be obtained. Fortunately, most organizations maintain their own software metrics repositories for each project (Fenton & Peeger, 1997). Among the repositories, many projects have similar reliability requirements. This paper has thus uniquely addressed a question: can the combination of multiple datasets with similar project characteristics help project managers achieve more robust and accurate patterns?

In this study, the technique applied to build SQC models is Genetic Programming (GP). It has been successfully applied in a large number of fields, such as circuit design (Dastidar, Chakrabarti, & Ray, 2005; Chang, Hou, & Su, 2006), data mining (Folino, Pizzuti & Spezzano, 2006; Wong & Leung, 2000), robotic control (Banzhaf, 1997), bioinformatics (Handley, 1995), and picture generation (Gritz & Hahn, 1997). The advantage of GP is that it can be used to discover solutions without predefining the size, shape and structure of those solutions. Moreover, as a parallel search-based technique, GP is a powerful tool for multi-objective optimization problems, and a valuable alternative to traditional statistical and data mining methodologies.

This paper describes the construction and comparison of two GP-based classifiers for SQC models based on seven NASA's projects implemented with different languages by different teams. One classifier uses data from a single data source in the training process, while the other classifier increases the volume of data by combining different data sources. Both classifiers are evaluated and compared by applying the statistical paired t-test. The results demonstrate that using data from multiple software projects with similar backgrounds significantly improves the accuracy and robustness of SQC models.

Since the SQC modeling is to discover faulty software modules, and the models are only available after coding is done, the comparisons between it and UML, or component-based software engineering approaches are not appropriate and therefore, not provided in this paper. However, SQC modeling can be integrated into other software engineering approaches (Ashley, Meehan, & Carr, 2005), or provide guideline to select highly reliable software components.

FUNDAMENTALS

SOFTWARE QUALITY CLASSIFICATION MODELING

SQC modeling focuses on discovering reliability patterns by classifying software modules into two classes: high-risk and low-risk. An SQC model is constructed from characteristic information extracted from an existing project, where its defect characteristic is known. Project managers can then apply the model to predict reliability of similar projects where defects are unknown. The characteristic information, such as the software product and process metrics, are usually collected by software tools and represented as a data structure. The structure used to train the SQC model is called a fit dataset; and the structure used to validate the SQC model is called a test dataset. SQC models can guide project managers to allocate available resources, and define more thorough test plans. Therefore, more successful software development can be achieved (Edwards & Steinke, 2007).

SQC modeling always misclassifies some modules due to data quality and model adequacy (Witten & Frank, 2005). A Type I error occurs when a model misclassifies a low-risk module as high-risk. A Type II error occurs when a model misclassifies a high-risk module as low-risk (Khoshgoftaar & Allen, 2000). The penalties for different misclassifications are not the same in software engineering practice. When a Type I error occurs, the cost is wasted resources trying to improve a high quality module. However, when a Type II error occurs, the cost could be very expensive even unaffordable, since a poor quality module could cause the whole system to fail.

The performance of an SQC model is measured by the Expected Cost of Misclassification (ECM) which is defined as:

$ECM = C_I N_I + C_{II} N_{II}$ (Khoshgoftaar and Allen, 2000).

Where C_I is the cost of a Type I misclassification, C_{II} is the cost of a Type II misclassification. N_I is the number of Type I errors, and N_{II} is the number of Type II errors. In order to simplify the discussions, we define $c = C_{II} / C_I$. The value of c is selected to achieve the project's preferred balance between the two misclassification rates.

SOFTWARE DATASETS

Seven datasets are extracted from seven NASA software systems. All are high assurance and complex real-time systems. The datasets include software measurement data and associated defect data collected at the function level (Khoshgoftaar & Seliya, 2004). The reliability of a module is described by whether or not the module has any defects, or by the number of defects in the module. Fenton and Peeger (1997) provide a complete description of the datasets. In order to discover the reliability patterns behind the datasets, thirteen common primitive software metrics are selected, normalized, and scaled (Khoshgoftaar & Rebours, 2004): Cyclomatic Complexity, Essential Complexity, Design Complexity, Loc Code And Comment, Loc Total, Loc Comment, Loc Blank, Loc Executable, Unique Operators, Unique Operands, Total Operators, Total Operands, and Branch Count. For the classifier which uses multiple projects in the training process, two important factors are available and need to be considered: whether an object-oriented language is used or not, and the size of the project. Therefore, two corresponding independent variables are added in the second classifier. The dependent variable is whether the module is high-risk or low-risk. Table 1 summarizes the properties of the seven datasets referred to as JM, KC1, KC2, KC3, CM, MW, and PC, respectively.

Table 1: Summary of the Software Data Repositories.

Dataset	NFP	FP	Total	Language	Size
JM	7163	1687	8850	C	Large
KC1	1782	325	2107	C++	Medium
KC2	414	106	520	C++	Small
KC3	415	43	458	JAVA	Small
CM	457	48	505	C	Small
MW	372	31	403	C	Small
PC	1031	76	1107	C	Medium

GENETIC PROGRAMMING

Genetic Programming imitates the Darwinian principle of survival and reproduction of the fittest individuals (Banzhaf et al., 1998; Koza, 1992). Each GP run may require several hundred or several thousand generations depending on the complexity of a problem. Each generation may contain thousands of individuals; each is an SQC model in our study. A fitness function, which evaluates the performance of individuals, assigns a value to each model, indicating how well the model solves the problem. A model with better performance is given a higher probability to produce offspring and pass its good substructure to the next generation. After a certain number of generations, the best model in a GP run represents the reliability pattern we are looking for based on the given fit dataset (Khoshgoftaar, Liu, & Seliya, 2004; Banzhaf, 1998). A GP process is described step by step below:

Initialization: GP generates the first population in the problem domain randomly. The individuals in the first population have extremely poor fitness except when the problem is so simple that it can be solved by a randomly generated solution. The first population can be generated using the half-and-half method: 50% of individuals in the first population are generated using the full method, where the individuals have to reach the specified maximum depth of trees. Another 50% of individuals are produced using the grow method where the individuals can have various depth.

Evaluation: After the first population has been generated, each individual is evaluated and assigned a fitness value by a fitness function using a set of fitness cases, which is the fit dataset in this study. Two measures of fitness used in this study are raw fitness and adjusted fitness (Koza, 1992). The raw fitness uses the natural terminology of a problem to express the fitness. It generally measures the amount of errors in an individual's attempted solution. In our research, it is computed as the total cost of misclassification. Adjusted fitness is computed as below.

$$a(i,t) = \frac{1}{1 + r(i,t)}$$

Where *i* is the ith generation, *t* is the *t*th individual, and *r(i, t)* is the raw fitness. The reason to use the adjusted fitness is that it can exaggerate small difference when the raw fitness approaches zero.

Selection: After the fitness of each individual is calculated, a selection algorithm is carried out to select individuals. An individual with better performance has higher possibility to be selected by the selection algorithm. The widely used tournament selection is applied in our study. The tournament chooses two or more models randomly. The model with the highest fitness will win and be applied to the following breeding process. The benefits of applying this method include accelerating the evolution process, paralleling the competition, and eliminating the centralized fitness comparison among the individuals.

Breeding: The breeding process contains three operations: crossover, mutation, and reproduction. The process starts with choosing a breeding operation randomly, then one or two individuals are selected based on the operation.

The crossover operation selects two individuals, randomly chooses a crossover point on each individual, exchanges the substructure below the points, and creates two new offspring individuals. The mutation operation chooses a random point in one selected individual, removes the substructure from that point, and inserts a randomly generated structure. The reproduction operation generates a new individual by coping the selected individual. All offspring generated from these operations are sent to the next generation.

After breeding, the new population replaces the old one. Then GP measures the fitness of the individuals in the new generation, selects the fitter ones, and repeats the breeding process. The whole process is repeated until the terminating conditions are satisfied. The terminating conditions could be either that an individual is found to solve the problem, or that the maximum number of generations is reached.

FITNESS EVALUATION

Two fitness functions are defined to evaluate model performance in the GP-modeling process. The first objective in this study is the accuracy of classification. Hence, ECM is used as the first and more important fitness function to evaluate performance of each SQC model. In order to shorten the running time and simplify SQC models, the size of the model is selected as the second fitness function. Namely, if two models have the same ECM values, then the smaller one indicates better performance. In this study, we empirically select 10 as a threshold to avoid losing major diversity in the early generations of a run. The effectiveness of this fitness function has been verified by Khoshgoftaar, Liu, & Seliya (2004).

METHODOLOGIES

The size and quality of data used by data mining algorithms often have strong implications for obtaining realistic assessments of predictive accuracy of generated models. Determining the adequacy of the datasets is difficult. In general, the larger volume of fit dataset is recommended to use for building more robust and accurate models except that the running time becomes unaffordable. Therefore, Classifiers 1 and 2 carry out the GP method introduced above to build SQC models, but they are using different volumes of training data. Specifically, classifier 1 uses the data from one software project to train the models, while classifier 2 is trying to increase the size of fit dataset by empirically exploring the homogeneity of data coming from different software projects.

CLASSIFIER 1

Classifier 1 contains seven experiments, each of which uses one of the seven datasets to train SQC models. In each experiment, 100 runs are performed and 100 models, each of which is the best model in a run, are recorded. Among the 100 models, the top three models are chosen to classify the modules in test datasets. If the difference between the performances of two models is too small, i.e., <0.5%, then the model size is used as a criterion for selection.

The independent variables used in Classifier 1 are thirteen primitive software metrics introduced in the Software Datasets section. The dependent variable identifies whether the module is high-risk or low-risk. When one experiments ends, the top three models are recorded and applied to each of the remaining six test datasets. The results show how well the models can predict risks of similar or future projects.

CLASSIFIER 2

Classifier 2 explores the homogeneity of data from different software projects by combining multiple datasets with similar project characteristics. It is to find more available data to build robust and accurate models. The modeling building process is the same as classifier 1 except that the fit dataset combines six of the seven datasets to train SQC models, leaving the remaining one as the test dataset.

As mentioned in the previous section, two additional independent variables are added during the training process. One is determined by whether an object-oriented language is used or not. If the programming language is C++ or JAVA, then the value of the variable is one; otherwise, the value of the variable is zero. The other categorizes the projects into three types-small, medium, and large-based on the number of modules in each project. In this study, a project is considered small if its number of modules is less than 1000, medium if its number of modules is in the range from 1000 to 5000, or large if its number of modules in above 5000. The last column of Table 1 shows the category of each project. It is worth mentioning a special case where the fit dataset includes KC1, KC2, KC3, CM, MW, and PC; because none of these fit datasets are large, the test dataset JM is re-classified as medium.

Table 2: Parameters list.

Pop Size	1000
max generations	50
random seed	3
output.basename	mutl1
output.bestn	1
init.method	half and half
init.depth	2-6
max depth	10
breed phases	3
breed(1).operator	crossover, sele=tournament
breed(1).rate	0.6
breed(2).operator	reproduction, sele=tournament
breed(2).rate	0.1
breed(3).operator	mutation, sele=tournament
breed(3).rate	0.3
function set	+,-,*,/,sin,cos,exp,log,GT, VGT

CASE STUDIES

PARAMETER LIST

The genetic programming tool known as lilgp1.01, developed by Douglas Zongker and Bill Punch of Michigan State University, is used in this study. The parameters are listed in Table 2. The values of the first eight parameters follow those provided by Koza (1992). The operators of breeding and the function set use the same values provided from the previous experiments (Khoshgoftaar, Liu, & Seliya, 2004). In addition, the maximum number of generations for Classifier 1 is 50, while it increases to 100 for Classifier 2 since multiple datasets are used in the training process.

EXPERIMENT RESULTS FOR CLASSIFIER 1

Results generated by Classifier 1 are shown in Tables 3 and 4. Table 3 shows the average results of the top three models for each fit dataset. The first column shows the names of the fit datasets. The second, third, and fourth columns indicate the Type I, Type II, and overall error rates, respectively. For example, the first row indicates the average Type I and Type II error rates of the top three models built based on PC are 22.50% and 22.81%. Namely, 22.50% low-risk modules are misclassified as high-risk modules, and 22.81% high-risk modules are misclassified as low-risk modules. The overall misclassification rate is 22.52%. Table 4 shows the average results achieved for each test dataset. It is worth mentioning that given a test dataset, each of the remaining six datasets can be used to train the SQC models and generate three top SQC models. Therefore, eighteen models are used to calculate the average results in Table 3. For example, when PC is chosen as the test dataset, the eighteen models obtained from six datasets, JM, CM, KC1, KC2, KC3, and MW, were applied to PC. The average Type I and Type II error rates are 38.70% and 36.77%, and the overall misclassification rate is 38.57%.

Table 3: The Average Results of Applying Classifier 1 on Fit Datasets.

Dataset	Error Rates		
	Type I	Type II	Overall
PC	22.50%	22.81%	22.52%
JM	40.39%	25.98%	37.64%
CM	28.01%	4.17%	25.74%
MW	18.46%	10.75%	17.87%
KC1	29.24%	18.05%	27.51%
KC2	18.20%	15.72%	17.69%
KC3	15.10%	13.95%	14.99%
Average	24.56%	15.92%	23.42%

EXPERIMENT RESULTS FOR CLASSIFIER 2

Each experiment of Classifier 2 selects three top models and applies them to the remaining test dataset. Since results coming from a small number of models might introduce more bias than those coming from a large number of models, each experiment is repeated three times. As a result, nine models are chosen and averaged. Table 5 shows the average results of the nine best models. The first column indicates which dataset is excluded from the seven datasets during the training process. The second, third and fourth columns indicate the Type I, Type II, and overall error rates, respectively. For example, the value, *exceptPC,* in the second row and first column indicates that dataset PC is excluded and the remaining six datasets are used to train the models. The resulting Type I, Type II, and overall error rates are 22.50%, 22.81%, and 22.52%, respectively.

Table 6 lists the results when these models are applied to the corresponding test datasets. The first column indicates which dataset is used as the test dataset. For example, the value PC in the second row of the first column indicates the results in the second row are achieved when applying the nine models obtained based on the datasets, JM, CM, KC1, KC2, KC3, and MW. In this case, Type I, Type II, and overall error rates are 44.13%, 15.79%, and 42.19%, respectively.

Table 4: The Average Results of Applying Classifier 1 on Test Datasets.

Dataset	Error Rates		
	Type I	Type II	Overall
PC	38.70%	36.77%	38.57%
JM	29.45%	51.46%	33.65%
CM	38.73%	37.04%	38.57%
MW	40.52%	32.62%	39.91%
KC1	25.91%	45.68%	28.96%
KC2	34.08%	34.70%	34.21%
KC3	26.37%	41.47%	27.79%
Average	33.40%	39.96%	34.52%

Table 5: The Average Results of Applying Classifier 2 on Fit Datasets.

Dataset	Error Rates		
	Type I	Type II	Overall
exceptPC	22.50%	22.81%	22.52%
exceptJM	40.39%	25.98%	37.64%
exceptCM	28.01%	4.17%	25.74%
exceptMW	18.46%	10.75%	17.87%
exceptKC1	29.24%	18.05%	27.51%
exceptKC2	18.20%	15.72%	17.69%
exceptKC3	15.10%	13.95%	14.99%
Average	24.56%	15.92%	23.42%

COMPARISONS BETWEEN TWO CLASSIFIERS

Table 7 shows the Type I, Type II errors, and Normalized Expected Cost of Misclassification (NECM), which is defined as: $NECM = ECM / N$, where N is the number of modules in the dataset. Because all of the seven software projects are safety-critical systems, the cost of a Type II error is considered much higher than that of a Type I error. Therefore, this study only reports cases where the cost of a Type II error is 15 times higher than the cost of Type I error. Specifically, the NECM values at c = 15, 20, and 25 are applied to compare the performances of Classifier 1 and Classifier 2. Figure 1 takes a closer look at the performances of the two classifiers. In general, the lower value of NECM indicates the lower misclassification costs, therefore, representing better model performance. The statistic paired t-test (Miller, 1986) is performed among these results. The concentration of paired t-test is the difference between the paired data. Each pair is the two NECM values generated by Classifier 1 and Classifier 2 at each cost ratio for each dataset. For example, the two NECM values obtained from Classifier 1 and Classifier 2 at $c = 15$ for JM are a pair. Table 8 lists the statistic paired t-test results calculated at 95% confidence interval. It contains three parts, the statistical results for Classifier 1, Classifier 2, and the differences between the paired data. The row, AAD, lists Average Absolute Deviation from Median; and the last row, p-value, lists the probability that the actual mean difference is consistent with zero. The low probabilities of rejecting differences (0.033 for $c = 15$, 0.038 for $c = 20$, and 0.043 for $c = 25$) indicate that combining different software repositories with similar project background can increase the amount of useful information. Thus, more accurate and robust reliability patterns can be discovered and used for project managers.

Table 6: The Average Results of Applying Classifier 2 on Test Datasets.

Dataset	Error Rates		
	Type I	Type II	Overall
PC	44.13%	15.79%	42.19%
JM	21.13%	58.45%	28.25%
CM	34.79%	20.83%	33.47%
MW	37.90%	25.81%	36.97%
KC1	31.03%	27.69%	30.52%
KC2	21.01%	23.58%	21.54%
KC3	22.41%	32.56%	23.36%
Average	30.34%	29.24%	30.90%

Table 7: The Comparison Results of Two Classifiers on the Test Dataset.

Dataset	Classifier	Type I	Type II	c=15	c=20	c=25
KC1	Classifier 1	25.91%	45.68%	1.276	1.628	1.981
	Classifier 2	31.03%	27.69%	0.903	1.117	1.33
KC2	Classifier 1	34.08%	34.70%	1.332	1.686	2.04
	Classifier 2	21.01%	23.58%	0.889	1.129	1.37
KC3	Classifier 1	26.37%	41.47%	0.823	1.018	1.212
	Classifier 2	22.41%	32.56%	0.662	0.815	0.968
JM	Classifier 1	29.45%	51.46%	1.71	2.2	2.691
	Classifier 2	21.13%	58.45%	1.842	2.399	2.956
CM	Classifier 1	38.73%	37.04%	0.879	1.055	1.231
	Classifier 2	34.79%	20.83%	0.611	0.71	0.809
PC	Classifier 1	38.70%	36.77%	0.739	0.865	0.992
	Classifier 2	44.13%	15.79%	0.573	0.628	0.682
MW	Classifier 1	40.52%	32.62%	0.75	0.876	1.001
	Classifier 2	37.90%	25.81%	0.648	0.747	0.846

CONCLUSION

In this study, data repositories from seven NASA software projects are applied to investigate the performances of two GP-based classifiers. These classifiers train the SQC models using data extracted from different projects. The results demonstrate that combination of data from several software repositories with similar characteristics can significantly improve the robustness and accuracy of SQC models, even though different languages and various methods are used in these software projects. Further research will focus on finding new methods to improve performance of SQC modeling based on multiple software projects.

Table 8: The Results of Paired T-Test between Classifier 1 and Classifier 2.

Classifier 1	**c=15**	**c=20**	**c=25**
Mean	1.07	1.33	1.59
Std	0.372	0.511	0.651
Highest	1.71	2.2	2.691
Lowest	0.739	0.865	0.992
Median	0.879	1.05	1.23
AAD	0.287	0.394	0.501
Classifier 2	**c=15**	**c=20**	**c=25**
Mean	0.875	1.08	1.31
Std	0.446	0.615	0.793
Highest	1.84	2.4	2.96
Lowest	0.573	0.628	0.682
AAD	0.257	0.366	0.507
Difference	**c=15**	**c=20**	**c=25**
Mean	0.197	0.255	0.279
Std	0.19	0.255	0.288
Highest	0.443	0.557	0.651
Lowest	-0.132	-0.199	-0.265
AAD	0.136	0.183	0.197
p-value	0.033	0.038	0.043

REFERENCES

Ashley, N. W., Meehan, T. E., & Carr, N. (2005). UML Activity Diagram Semantics and Automated GUI Prototyping. *Journal of Information Technology and Information Management*. 14(3), 43-54.

Banzhaf ,W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic Programming: An Introduction On the Automatic Evolution of Computer Programs and its Application*. PWS Publishing Company, New York, NY.

Banzhaf, W., Nordin, P., & Olmer, M. (1997). Generating adaptive behavior for a real robot using function regression within genetic programming. *Genetic Programming 1997: Proceedings of the Second Annual Conference,* Stanford, CA, 35-43.

Briand, L. C., Melo, W., & Wuest, J. (2002). Assessing the applicability of fault-proneness models across object-oriented software projects, *IEEE Transactions on Software Engineering, 28(7)*, 706-720.

Chang, S. J., Hou, H. S., & Su, Y. K. (2006), Automated passive filter synthesis using a novel tree representation and genetic programming, *IEEE Transactions on Evolutionary Computation, 10(1),* 93-100.

Dastidar, T. R.,. Chakrabarti, P. P., & Ray, P. (2005). A synthesis system for analog circuits based on evolutionary search and topological reuse, *IEEE Transaction on. Evolution. Computation., 9(2),* 211–224.

Edwards, J. N, & Steinke, G. (2007). The Development of a Thorough Test Plan in the Analysis Phase leading to more Successful Software Development Projects, *Journal of Information Technology and Information Management*. 16(1), 65-72.

Fenton, N. E., & Peeger, S. L. (1997). *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, Boston, MA, 2nd edition.

Fenton, N., & Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex software system, *IEEE Transactions on Software Engineering, 26(8)*, 797-814.

Folino, G., Pizzuti,C., & Spezzano, G. (2006). GP ensembles for large-scale data classification *IEEE Transactions on Evolutionary Computation, 10(5),* 604-616.

Gritz, L., & Hahn, J. K. (1997). Genetic programming evolution of controllers for 3-D character animation. *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford, CA, 139-146.

Handley, S (1995). Predicting whether or not a nucleic acid sequence is an e coli promoter region using genetic programming. In *Proceedings of the First International Symposium on Intelligence in Neural and Biological Systems*, Herndon, VA, 122-127.

Khoshgoftaar, T. M., & Allen, E. B. (2000). A practical classification rule for software quality models. *IEEE Transactions on Reliability, 49(2)*, 209-216.

Khoshgoftaar, T. M., & Seliya, N. (2004). The necessity of assuring quality in software measurement data. In *Proceeding: 10th IEEE International Symposium on Software Metrics*, Chicago, IL, 119-130.

Khoshgoftaar, T. M., & Rebours, P. (2004). Generating multiple noise elimination filters with the ensemble-partitioning filter. In *Proceeding: 2004 IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, 369-375.

Khoshgoftaar, T. M., Liu, Y., & Seliya, N. (2004). A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation, 8(6),* 593-609.

Khoshgoftaar, T. M., Seliya, N., & Herzberg, A. (2005). Resource-oriented software quality classification models. *Journal of System and Software,76(2),* 111-126.

Kitchenham, B. (1996). *Software Metrics Measurement for Software Process Improvement*. Blackwell, Cambridge MA.

Lyu, M. R. (1996). *Handbook of Software Reliability Engineering*. IEEE Computer Society and McGraw-Hill Press, Cambridge, MA.

Miller, R. G. Jr. (1986). *Beyond ANOVA, Basics of Applied Statistics.* John Wiley & Sons, New York, NY.

Nagappan, N., Ball, T., & Murphy, B. (2006). Using historical in-process and product metrics for early estimation of software failures, *Proceedings of the 17th International Symposium on Software Reliability Engineering,* Washington, DC, 62-74.

Post, G.V, & Kagan, A. (2005). Systems Development Tools and the Relationship to Project Design: Cost and Budget Implications, *Journal of Information Technology and Information Management*. 14(1), 1-14.

Sampson, S. E, & Hulet, K. (2003) An Empirical Model of Price and Quality Effects of Ecommerce, *Journal of Information Technology and Information Management*. 12(2), 1-16.

Tian, J., & Palma, J. (1998). Analyzing and improving reliability: A tree-based approach. *IEEE Software, 15(2),* 97-104.

Witten, I., & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques,* Morgan Kaufmann, San Francisco, CA.

Wong, M. L. & Leung, K. S. (2000). *Data Mining Using Grammar Based Genetic programming and Applications(3),* Kluwer Academic, Boston, MA.

ACKNOWLEDGEMENTS

We thank Dr. Taghi M. Khoshgoftaar for providing data in this study.