METHODOLOGY FOR IDENTIFYING FACTORS THAT IMPACT SOFTWARE QUALITY[1]

Vishnuprasad Nagadevara, Indian Institute of Management Bangalore, India

ABSTRACT

Software quality has become an important concern in our daily lives. Understanding the factors that influence software quality is crucial to the software industry. An improved understanding of software quality drivers will help software engineers and managers make more informed decisions in controlling and improving the software process. While many of the studies on software quality have focused on the measurement aspects of software quality, very few have analyzed the factors that could influence the quality itself. One of the main reasons for this is the fact that the quality is measured in terms of ordinal or nominal data while the factors which are numerous, are measured in terms of indices or real numbers. Traditional techniques such as regression or analysis of variance which are generally used to measure the impact of different variables on the quality fail in such a situation. On the other hand, techniques such as artificial neural networks, logistic regression or classification trees work better with dependent variables which are categorical or ordinal. This paper demonstrates the use of these techniques to identify the factors that effect software quality and attempts to rank these factors in the order of their importance.

Keywords: *Software Quality, Artificial Neural Networks, Classification Trees, Logistic Regression*

1. INTRODUCTION

Software has become an integral part of our lives in recent days. Most consumer appliances, including communication devices and automobiles, have significant software components. The most complex systems, such as airplane flight control or nuclear power plants, depend critically upon the reliability of software. Today, the need to understand, control, and design quality software is of utmost importance [Gibbs, 1994]. While different metrics are used for measuring software quality, there are five disciplinary approaches widely used in quality definition. These are Transcendent, Product-Based, User-based, Manufacturing-Based and Value-Based [Garvin, 1988]. Investigations by Usrey and Dooley (1996) suggest that there are several dynamic elements that effect quality attitudes. Powerful mathematical tools exist for modeling dynamic systems. Software quality Models featuring interactions, delays, and feedback often produce unique and useful predictions.

Software testing itself is an integral part of assuring quality. National Institute of Standards & Technology (NIST) estimated that the national annual cost of an inadequate infrastructure for software testing in the USA is $59.5 billion. The same study estimated that the potential cost reduction from feasible infrastructure improvements is $22.2 billion. This represents about 0.6 and 0.2 percent of the U.S.'s $10 trillion dollar GDP respectively. Software developers accounted for about 40 percent of total impacts, and software users accounted for the about 60 percent [NIST, 2002].

Software quality assessment and quality assurance are developing into complex subjects. The most traditional program analysis techniques are McCabe Cyclomatic Complexity (McCabe, 1976), Halstead Software Science (Halstead, 1977) volumetric techniques, and a wide range of specifically tailored structural and graph-based approaches developed over the last two decades. Martin and Shafer (1996) used their own assessment methodology and its support tools to assess over 31 million lines of code in over four dozen languages.

The best known and most thoroughly studied of what are classified as composite measures of complexity emerge from Halstead's theory of software science [Halstead 1977]. Halstead argued that algorithms have measurable characteristics analogous to physical laws. His model is based on four different parameters: the number *of* distinct operators (instruction types, keywords, etc.) in a program, called **nl;** the number *of* distinct operands (variables and constants), **n2;** the total number of occurrences of the

[1] Journal of Academy of Business and Economics, Vol 8, no. 2, 2008, pp145-151

operators, **N1;** and, the total number of occurrences of the operands, **N2.** The sum of **nl** and **n2** is denoted as **n** while the sum of **N1** and **N2** is called **N.** From those four counts, a number of useful measures can be obtained. Many researchers have used these measurable characteristics developed by Halstead in software quality.

Khaddaj and Horgan [2004] focused on software quality factors that should be taken into account in very large information systems. Such systems will require a high degree of parallelism and will involve a large number of processing elements. They identified the metrics and measurement approaches that can be applied for sequential and parallel/distributed architectures and investigated a number of factors which are relevant to the parallel class systems. In such a system, many elements can fail which can have major impact on the system's performance, and therefore it affects the costs and benefits. They showed that portability and usability are other major problems that need to be taken into account when considering all the relevant factors that affect quality for such environments.

Heemstra and Kusters [2002] focused on the "soft" factors that could influence the software quality. Much discussion about software quality focuses on the technical aspects of the development process, with little regard for "soft" factors such as motivation, commitment, organizational culture, and other drivers important to achieving high product quality. They have presented a framework that can be used as a tool for rank ordering software quality drivers.

Understanding the factors that influence software quality is crucial to the software industry. An improved understanding of software quality drivers will help software engineers and managers make more informed decisions in controlling and improving the software process. Paulk (2006) found that program size, (empirically measured) programmer ability, and disciplined processes significantly affect software quality. He also found that factors frequently used as surrogates for programmer ability such as years of experience, and technology variables such as programming language, do not significantly impact software quality, although they may affect other important software attributes such as productivity.

While many of the studies on software quality have focused on the measurement aspects of software quality, very few have analyzed the factors that could influence the quality itself. One of the main reasons for this is the fact that the quality is measured in terms of ordinal or nominal data while the factors which are numerous, are measured in terms of indices or real, quantifiable numbers. Traditional techniques such as regression or analysis of variance which are generally used to measure the impact or influence of different variables on the quality fail in such a situation. On the other hand, techniques such as artificial neural networks, logistic regression or classification trees work better with dependent variables which are categorical or ordinal in nature.

This paper attempts to develop a methodology for identifying various factors that influence the software quality. Most of the factors that are considered for the analysis are those which are measured routinely in the development of software.

The objectives of the study are to (a) identify the appropriate techniques that could be used to isolate the factors that influence the software quality, (b) use the techniques so identified to rank these factors in terms of their relative importance in assuring software quality and (c) identify the factors that are likely to be critical in assuring software quality.

2. METHODOLOGY

Data with respect to a total of 7800 modules spread over a number of software projects was obtained for the analysis. These modules are categorized into "good" or "bad" modules based on the errors contained in the modules. There were 5362 "good" modules and 2438 "bad" modules. The entire data set of 7800 modules was divided randomly into two sets, one for training the model and the other for testing the model. The training dataset consisted of 5900 module and the rest were in the testing dataset.

The dataset contained information on various variables for each module. A total of 40 such variables were identified for the purpose of the analysis. Eight of these variables were the Halstead measures

namely content, difficulty, effort, error estimate, length, level, time and volume. Other variables are numerical in nature which include number of lines of code, number of branches, blank lines, function calls, code-cum-comment lines, number of parameters in the module, number of comment lines, number of conditions, number of decision points, number of transfer of controls and number of nodes. Other variables included cyclomatic complexity, design complexity, cyclomatic density, design density, essential complexity and density, global data complexity and density, as well as number of operators and operands. The number of unique operands and operators in each module were also counted. Certain other variables are derived from the above variables. All the variables were normalized before carrying out the analysis.

The following methodology was adopted for identifying the important factors that affect the software quality. Three different techniques namely Artificial Neural Networks, Classification Trees and Logistic regression were used as training models for classifying each of the modules as "good" or "bad" module. Normally, the purpose of such classification models is to be able to predict the future models with the likelihood of "good" or "bad" categories. In this paper, the classification models are built and the factors that are important for making the appropriate prediction are identified. These factors or variables play an important role in prediction of "good" or "bad" modules and consequently these are the variables that have an impact of the software quality. Initially, the prediction models are built using the training dataset and the effectiveness of these models is tested on the testing dataset. This is a standard procedure to make sure that the models built on the training dataset are replicable and not a result of any accidental relationships. A brief description of the three techniques used for the analysis is presented below.

Artificial Neural Networks

The artificial neural networks (ANN) are based on the concepts of the human (or biological) neural networks consisting of neurons, which are interconnected by the processing elements. The ANNs are composed of two main structures namely the nodes and the links. The nodes correspond to the neurons and the links correspond to the links between neurons. The ANN accepts the values of inputs into its input nodes (or input layer). These values are multiplied by a set of weights and added together to become inputs to the next set of nodes to the right of the input nodes. This layer of nodes is referred to as the hidden layer. Many ANNs contain multiple hidden layers, each feeding into the next layer. Finally, the values from last hidden layer are fed into an output node, where a mapping or thresholding function is applied and the prediction is made. The ANN is created by presenting the network with inputs from many records whose outcome is already known. For example, the data on different variables of the first software module (first record) are inputted into the input layer. These values are fed into the hidden layer and after processing the prediction is made at the output node. If the prediction made by the ANN matches with the actual known status of the module ("good" or "bad"), then the prediction is good and the ANN proceeds to the next record. If the prediction is wrong, then the extent of error is apportioned back into the links and the hidden nodes. In other words, the values of the weights at each link are modified based on the extent of error in prediction through a process called backward propagation. The artificial neural networks are found to be effective in detecting unknown relationships. ANNs have been applied in many service industries such as health to identify the length of stay and hospital expenses (Nagadevara, 2004), and for predicting the categories of the members of the loyalty programmes (Nagadevara 2005).

Logistic Regression

Logistic regression is a specialized form of regression used to predict and explain a categorical dependent variable. It works best when the dependent variable is a binary categorical variable. The regression equation developed is very similar to a multiple regression equation with "regression-like" coefficients which explains the impact of each of the independent variable in predicting the category of the dependent variable. One special advantage of logistic regression is that it is not restricted by the normality assumption which is a basic assumption in the regression analysis. It can also accommodate non-metric variables such as nominal or categorical variables by coding them into dummy variables. Another advantage of logistic regression is that it directly predicts the probability of an event occurring. In order to make sure that the dependent variable, which is the probability, is bounded between zero and one, the logistic regression defines a relationship between the dependent and independent variables that

resembles an S-shaped curve. It uses an iterative process to estimate the "most likely" values of the coefficients. This results in the use of a "likelihood" function in fitting the equation rather than using the sum of squares approach of the regression analysis. The dependent variable is considered as the "odds-ratio" of a specific observation belonging to a particular group or category. In that sense, logistic regression estimates the probability directly. In order to get the best results from the logistic regression, it is important to have continuous variables as independent variables. It is also important to define the nominal variables appropriately, so that they are converted into the required number of dummy variables.

Classification Trees (C5.0)

C5.0 is one of the popular methods of building classification trees. The classification tree is built by splitting the observations at each node based on a single attribute or independent variable such that the resultant sub-groups are more homogenous with respect to the categories of dependent variable. If no split that could significantly reduce the diversity of the dependent variable at a given node could be found, the process of splitting is stopped and the node is labeled as a leaf node. When all the nodes become leaf nodes, the tree is fully grown. At the end of the construction of the tree, each and every observation has been assigned to a leaf node. Each leaf can now be assigned to a particular class or category and a corresponding error rate. The error rate at the leaf node is nothing but the percentage of misclassifications at the leaf node. The error rate for the entire tree is the weighted sum of the error rates of all the leaf nodes. In the case of C5.0 classification trees, information gain is used as the criterion for splitting the records at each node. Entropy is used to measure the information gain. This method can generate trees with variable number of branches at each node. For example, when a discrete variable is selected as an attribute for splitting, there would be one branch for each value of the attribute.

3. RESULTS AND ANALYSIS

As mentioned earlier, a binomial logistic regression equation is fitted with the category of the software module ("good" or "bad") as the dependent variable and all other variables as the independent variables. In fact, the dependent variable itself is the odds-ratio of the category of the module. Only those variables which are statistically significant have been included in the equation. There are 14 variables (factors) that are statistically significant. The coefficients of the equation along with the associated statistical tests are presented in Table 1. The model is applied to training data as well as testing data to measure the effectiveness of the regression equation in predicting the category of the module. These predictions are presented as the prediction matrix in Table 2.

Table 1. Regression coefficients of the Logistic Regression

Variable	B	Std. Error	Wald	df	Sig.	Exp(B)	95% C. I. for Exp(B)	
							Lower	Upper
Intercept	0.090	0.067	1.776	1	0.183			
Decision Points	-3.651	0.725	25.342	1	0.000	0.026	0.006	0.108
Essential Complexity	-0.827	0.238	12.035	1	0.001	0.438	0.274	0.698
Global Data Density	3.388	0.220	236.556	1	0.000	29.602	19.223	45.584
Halstead Difficulty	-0.503	0.062	65.709	1	0.000	0.605	0.536	0.683
Halstead Effort	-2.517	0.366	47.304	1	0.000	0.081	0.039	0.165
Blank Lines	-0.029	0.015	3.635	1	0.057	0.972	0.943	1.001
Comment Lines	0.938	0.174	28.951	1	0.000	2.555	1.816	3.597
Maintenance Severity	-0.290	0.047	37.334	1	0.000	0.748	0.682	0.821
Multiple Conditions	4.513	0.998	20.446	1	0.000	91.171	12.893	644.713
Number of Operators	0.593	0.177	11.259	1	0.001	1.809	1.279	2.557

Variable	B	Std. Error	Wald	df	Sig.	Exp(B)	95% C. I. for Exp(B)	
							Lower	Upper
Number of Operands	1.186	0.454	6.831	1	0.009	3.276	1.345	7.974
Unique Operands	1.474	0.167	77.745	1	0.000	4.366	3.147	6.059
Unique Operators	-0.351	0.074	22.533	1	0.000	0.704	0.609	0.814
No. of Parameters	-0.385	0.042	82.310	1	0.000	0.681	0.626	0.740

Table 2. Prediction matrix for the three models

		Prediction					
		Training Data			Testing Data		
	Category	Bad	Good	Total	Bad	Good	Total
	Logistic Regression						
Actual	Bad	882	993	1875	249	314	563
	Good	240	3785	4025	78	1259	1337
	Artificial Neural Network						
	Bad	1095	780	1875	330	233	563
	Good	308	3717	4025	99	1238	1337
	Classification Tree (C 5.0)						
	Bad	1606	269	1875	462	101	563
	Good	1151	2874	4025	419	918	1337

In the next step, an Artificial Neural Network was built using all the independent variables. The prediction matrix with respect to the Artificial Neural Network is also presented in Table 2. While the neural networks do not present any coefficients with respect to the independent variables, the software package calculates a sensitivity index which represents the relative importance of various factors. Table 3 presents the relative importance (sensitivity index) of the variables used in building the neural networks. It can be seen from Table 3 that 9 out of the top 15 variables are the same as the ones that were statistically significant in logistic regression.

Table 3. Relative importance of variables based on ANN

Variable	Sensitivity	Variable	Sensitivity	Variable	Sensitivity
Unique Operands	0.7294	Blank Lines	0.3415	No. of Operators	0.2360
Global Data Density	0.7244	Nodes	0.3157	Number of Operands	0.2214
Design Complexity	0.7114	lines	0.3007	Edges	0.1984
Lines of Code	0.7090	Halstead Difficulty	0.2946	Halstead Effort	0.1768
Executable code	0.7083	Halstead Level	0.2831	Cyclomatic Density	0.1519
Comment Lines	0.7004	Function Calls	0.2730	Halstead Volume	0.1405
Condition Modification	0.6946	Total Lines	0.2706	Halstead Error	0.0946
Multiple Conditions	0.6757	Halstead Length	0.2602	Maintenance Severity	0.0815
Conditions	0.5426	Decision points	0.2539	Branches	0.0329
Unique Operators	0.4514	Halstead Content	0.2465		
Essential Complexity	0.3933	No. of Parameters	0.2462		

Finally, a third classification technique namely Classification trees was applied to the same dataset. The prediction matrix of the classification tree is also presented in Table 2. The Classification Tree had 17 levels of which the top six are presented in Figure 1. Twelve variables that appear in the top six levels of the tree are same as those which were statistically significant in the logistic regression model. In other

words, all the three techniques used for prediction of the software quality had a large number of common variables influencing the prediction/classification of the software module. Thus, these are the variables that will have a significant impact on the software quality. The importance of these variables in software quality is reiterated by each of the techniques in the sense that there are a number of common variables that are significant in all the three techniques.

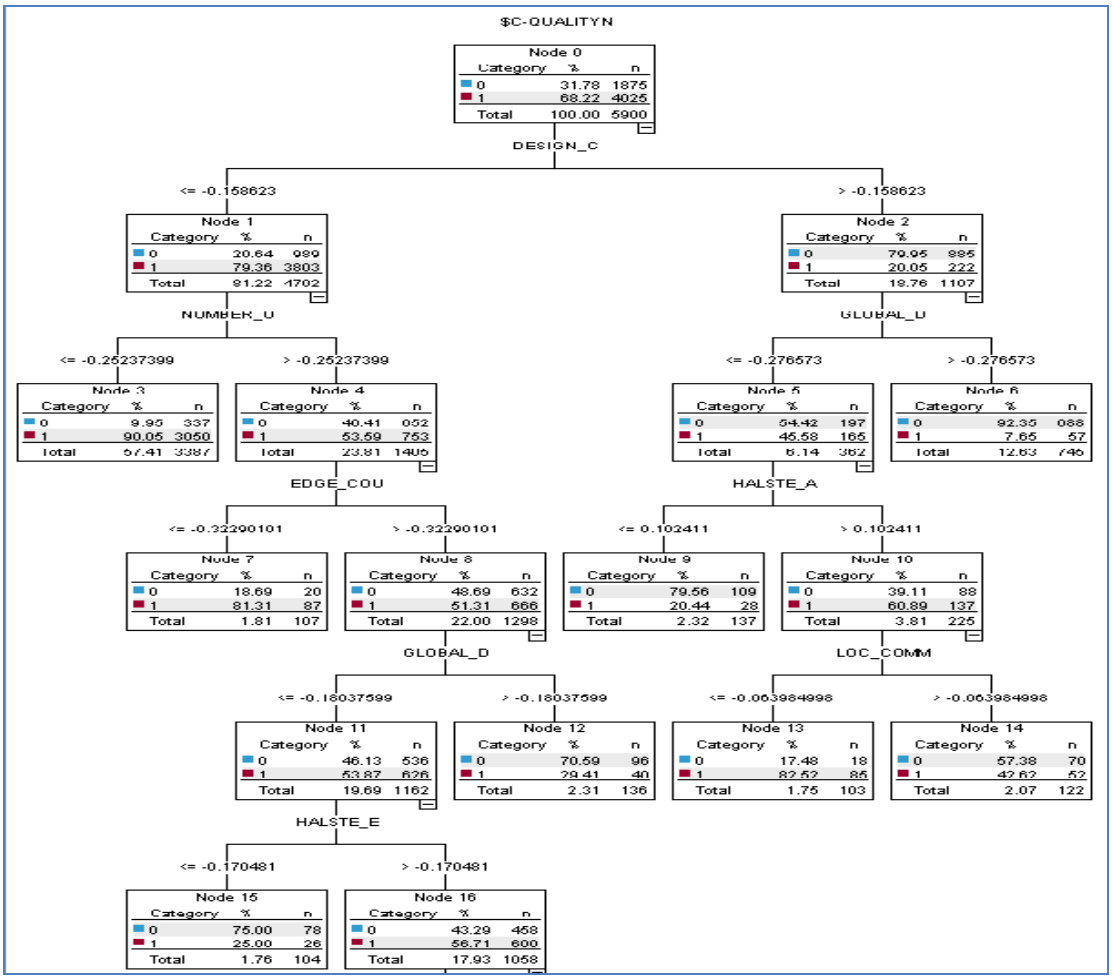

Figure 1. Classification Tree for predicitng the software quality

The results presented in Table 2 show that the prediction accuracies for both training data and the testing data are very similar indicating that the three models are stable and robust. Thus, these models could be applied for predicting the category of the modules based on different variables. There are some differences in the prediction accuracies of the three techniques used. Logistic regression is able to provide the best prediction with respect to the "good" modules while classification trees provide the best prediction for "bad" modules. The importance of each of the factors can be gauged based on the common occurrences of the variables across the three techniques used. Nine factors are identified as the critical factors influencing the software quality through this process. These are number of unique operands, number of unique operators, multiple conditions, global data density, number of comment lines, number of blank lines, Halstead difficulty, essential complexity and number of lines of code. Since these are identified as the critical factors, the software quality can be improved by paying more attention to these variables while developing the software modules.

4. SUMMARY AND CONCLUSIONS

As software is becoming an integral part of our lives, software quality is becoming more critical. While there have been a number of research studies dealing with software quality and its measurement, not

enough research had gone into identifying various factors that influence software quality. One of the primary reasons for this is that the quality is a nominal or ordinal concept which does not lend itself to techniques such as regression analysis. On the other hand, there are other techniques such as Artificial Neural Networks, Classification Trees and Logistic Regression which are routinely used for prediction of variables that are nominal. This paper has applied these three techniques to predict the software quality using large number of variables. By identifying the variables that are significant across all these three techniques, the study is able to isolate those variables which have a significant impact on the software quality. In other words, the software companies need to be more vigilant with respect to these variables as compared to the others. By paying more attention to these variables vis-à-vis other variables, the software companies will be in a position to improve the quality. It may be necessary for these companies to test the methodology and the results for different types of software modules specific to the company. Since the techniques used in this methodology are predictive in nature, a prediction about the possible outcome of the software module (either "good" or "bad") along with the associated probability can be made using the independent variables. Additional care can be taken in the development efforts based on this prediction. This is an additional benefit that arises out of this methodology.

5. REFERENCES

Gibbs, W.W., "Software's Chronic Crisis," *Scientific American*. September 1994, p. 86-95.

Halstead, M. H., "Elements of Software Science", New York: Elsevier North-Holland, 1977.

Heemstra, Fred J and R. J. Kusters, "Soft Factors Affecting Software Quality**",** *Software Quality Professional*, Vol. 5, No. 1, December 2002, pp. 20-29:

Khaddaj, Souheil and G Horgan, "The Evaluation of Software Quality Factors in Very Large Information Systems", Electronic Journal of Information Systems Evaluation Volume 7 Issue 1 (2004) 43-48

Martin, R. A. And Lawrence H. Shafer, "Providing A Framework For Effective Software Quality Measurement: Making A Science Of Risk Assessment, available at http://www.mitre.org/work/tech_transfer/pdf/risk_assessment.pdf last accessed June 11, 2008

McCabe, T. J., "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol SE2, No. 4, 1976, 308-320.

Nagadevara, Vishnuprasad (2004) *Application of Neural Prediction Models in Healthcare*, Proceedings of the 2nd International Conference on e-Governance, Nov 29 – Dec 1, 2004, Colombo, Sri Lanka, pp 139-148.

Nagadevara, Vishnuprasad (2005), Improving the Effectiveness of Hotel Loyalty Programmes through Data Mining, Proceedings of the International Conference on Services Management, March 11-12, 2005, New Delhi, pp70-74

National Institute of Standards & Technology, The Economic Impacts of Inadequate Infrastructure for Software Testing, RTI Project Number 7007.011, May 2002, available at http://www.nist.gov/director/prog-ofc/report02-3.pdf last accessed June 11, 2008

Paulk, M. C., "Factors Affecting Personal Software Quality", *Cross Talk - The Journal of Defense Software Engineering*, March 2006, pp 9-13.

Usrey, M., and K. Dooley, "The Dimensions of Software Quality," *Quality Management Journal*, 3(3), 1996: 67-86.

Author Profile

Dr. Vishnuprasad Nagadevara obtained his Ph D from Iowa State University, Ames Iowa. He is currently Professor in the Quantitative Methods and Information Systems Area at the Indian Institute of Management Bangalore. His current research interests are Data Mining, Application of OR Techniques to Management.