AN EXPLORATORY STUDY ON A LINEAR MODEL FOR MEASURING SOFTWARE QUALITY

Angel R. Puerta and Charles L. Carnal

Tennessee Technological University
Cookeville, Tennessee

Puerta, A.R., and Carnal, C. An Exploratory Study on a Linear Model for Measuring Software Quality. 1989 IEEE SouthEastCon, Columbia, South Carolina, April 1989.

Abstract

A pilot study was conducted to develop a linear regression model to measure software quality. The model incorporated four quality components representing efficiency, understandability, modifiability, and implementation of requirements. Halstead measures were used as regressors along with a programmer rating. The study employed a group of second semester FORTRAN programming students who implemented the same program individually. A series of standard tests were utilized to measure the quality components.

The results showed linear dependency of all quality components, except understandability, on Halstead metrics. The efficiency, modifiability, and requirements components were expressed using only Volume, Level, and Programming Time metrics. Thus, those three measures appear as the most promising for validation research. However, the best correlations for each quality component were given by the programmer rating suggesting that non-linear solutions might be more appropriate.

Introduction

Software reliability and software quality assurance are two aspects of software engineering that have lagged somewhat behind other parts of the development methodology. One of the reasons for this problem is the fact that there is no standard or concrete way of measuring the quality of a program. In this paper, software quality is defined as a relative measure of the degree to which a program is expected to satisfy its requirements, deliver usable services, and at the same time be concise, consistent, efficient, maintainable, portable, and understandable. This study concerned itself with exploring the possibility of using some well-known software metrics to construct a software quality metric.

The importance of obtaining such a metric is visible from two different stages of the software development cycle. In the pre-testing stage, a project manager could use quality metrics to determine the kind and level of testing that must be used. In the post-testing stage, quality metrics could be used to compare functionally similar programs. It would be safe to assume that, with the rising cost in software production and maintenance, the present trend towards automatic programming and reusability of software will be strongly encouraged. It would be crucial then to have a standard or universal way of assessing the quality of software.

Most work on this area up to now has dealt with identifying the factors that affect software quality. As Curtis has reported, Boehm, Brown, Kaspar, and Mc Call have developed models which intuitively cluster software characteristics related to quality [4]. However, these approaches fail to either rank or quantify the parameters included in the models. In addition, extensive work has been performed on various software metrics which measure diverse software characteristics such as complexity, error proneness, and reliability. Kafura [8], Shen [16], and Davis [6], have evaluated the merits and shortcomings of several of these metrics.

Many of the factors affecting software quality that have been identified by researchers can be seen in part as functions of the complexity and size of the program, and the capabilities of the programmers and managers. This will include, but is not limited to, testability, efficiency, legibility, and structuredness. There are a number of ways to quantify complexity in a program. The best known metrics which provide such feature are McCabe's [11] and Halstead's [7]. These metrics have been extensively validated and compared [1-4,9,12,14-15]. Other quality factors, like portability, can depend on hardware as well as software. Of course, not all factors can be maximized for any particular program and some factors can be of no significance for a particular application. In general, little progress has been made in quantifying these factors, establishing their relative importance in the overall quality picture, and identifying their interrelationships.

In our study, we explore the possibility that some relationship might exist between Halstead's measures, programmer ratings, and software quality. Halstead measures were chosen because research shows that they are among the most promising metrics available [6,8,16]. A group of second semester programming students were assigned the same program to be completed individually. The grade point average of each student was used as a programmer rating. The resulting programs were measured for completeness of requirements, efficiency, understandability, and modifiability against a series of standard tests. Then, multivariate linear regression was utilized to derive correlations and linear models using the programmer rating and Halstead metrics as regressors.

Halstead Measures

The best known and most thoroughly studied of what are classified as composite measures of complexity emerge from Halstead's theory of software science [7]. It is suggested to the interested reader that he refers to the original manuscript by Halstead for a good understanding of these metrics. 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 **n1**; the number of distinct operands (variables and constants), **n2**; the total number of occurrences of the operators, **N1**; and, the total number of occurrences of the operands, **N2**. The sum of **n1** 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. The number of bits required to specify the program is called the volume **V** of the program and is obtained through the equation

$$\mathbf{V} = \mathbf{N} \log 2\ \mathbf{n} \qquad (1)$$

The program level which is the difficulty of understanding a program is calculated by

$$\mathbf{L} = (2\mathbf{n}2)/(\mathbf{n1N2}) \qquad (2)$$

and the intelligence content of a program is given by

$$\mathbf{I} = \mathbf{L} \times \mathbf{V} \qquad (3)$$

In an attempt to include the psychological aspects of complexity in the measures, Halstead studied the cognitive processes related to the perception and retention of simple stimuli. Research by Stroud [17] had shown that the mean number of mental discriminations per second in an average human being, also called the Stroud number, is between 5 and 20. Halstead uses 18 as a reference point for his studies. In his model, the number of discriminations made in the preparation of a program, called effort, is given by

$$\mathbf{E} = \mathbf{V}/\mathbf{L} \qquad (4)$$

The programming time, **T**, is an estimate of the number of mental discriminations necessary to complete a program divided by the average number of discriminations per second or Stroud number, **S**. It is important to note that this estimate assumes that the programmer is devoting all of its discriminations to the programming task. Therefore, the estimate is a minimum value since, in reality, the programmer would use some mental effort on non-related tasks.

All of these measures are valid under the assumption that the program is "pure," i.e., free of so-called "poor programming practices." Halstead defines six classes of impurities, among them, synonymous operands, unfactored expressions, and common subexpressions. The complete description of these and other impurities is beyond the scope of this paper. However, for the programs used for this paper, all recognizable impurities were eliminated prior to obtaining the corresponding Halstead measures.

The Experiment

The four major components of the experiment setup were the study group, the program, the quality indexes, and the regression methodology.

The population studied consisted of a section of 43 second semester FORTRAN programming students. A total of 13 subjects were rejected for either not completing the assignment or turning in programs which were judged not to have been completed individually. The grade point average of the students in technical courses was taken as a programmer rating, **PR**.

The program to be completed required the students to read several lists of integers under a pre-defined criteria, merge, sort, and print the resulting lists. This program was chosen because, in spite of its small size, it offered a variety of ways to be implemented. The development of the program was divided into two phases. In the first phase, the program was coded and debugged until the first successful compilation. Thus, the program at such point was free of syntax errors. This stage would correspond in the usual software life cycle to the moment immediately prior to unit testing. After completion of the first phase, the Halstead measures **V**, **L**, **I**, **E**, and **T** were obtained. During the second phase, the subjects discovered and removed existing errors until the program was brought into full functionality. The amount of time used for debugging and testing purposes, **TDEB**, and the number of errors found, **ERR**, were recorded. These two measures were later used to validate the quality indexes calculated during the testing period.

Four independent quality indexes were computed using a battery of standard tests. A requirements index, **QR**, was given to reflect the number of requirements successfully implemented in each program. An efficiency index, **QE**, represented the c.p.u. time used by the program to run a standard set of data. An understandability index, **QU**, and a modifiability index, **QM**, were assigned subjectively by three distinct evaluators. Although such a methodology does not allow the repeatability necessary for validation research, it is appropriate for pilot or exploratory studies. The **QE** index was validated against **ERR** while **QM** was validated against **TDEB**.

Linear regression using least-square estimators was employed with each of the quality indexes to obtain the respective linear models. The variables **PR**, **V**, **L**, **I**, **E**, and **T** were used as regressors. The regression proceeded in the forward direction meaning that the most significant variable was added first to the model. The weight of each variable was given by its level of significance which had a lower limit of 0.5. Such a high limit is due to the exploratory nature of the study. A statistic developed by Mallows [5,10], **C(p)**, served to determine the ideal number of variables in the model. Graphing **p**, the number of variables in a model plus the intercept, against **C(p)** provides the ideal value when **p** first approaches **C(p)**. Thus, some variables which meet the level of significance criteria may be excluded from the model if the graph so determines it.

The Results

The main results derived were the four linear models which looked as follows:

$$\mathbf{QR} = a\mathbf{PR} - b\mathbf{L} + c\mathbf{T} - d\mathbf{I} + e \qquad (5)$$

$$\mathbf{QE} = a\mathbf{PR} + b\mathbf{T} - c\mathbf{E} - d \qquad (6)$$

$$\mathbf{QM} = a\mathbf{PR} - b\mathbf{V} + c\mathbf{T} - d\mathbf{E} + e \qquad (7)$$

$$\mathbf{QU} = a\mathbf{PR} - b \qquad (8)$$

The order of inclusion for the variables in each model is from left to right. The actual value of the coefficients is not relevant in this type of study but rather the sign of each parameter is of more interest.

The conclusiveness of the linearity of each model was determined using the **F** statistical value. All four models were above the threshold value, thus confirming the significance of their linearity. Each model, except **QU**, showed dependency on Halstead measures. The lack of correlation between the metrics and **QU** is probably due to the possibility of improving understandability, with for example indentation, without changing the number of operands and operators in the program.

The programmer rating consistently delivered the highest correlation with the quality indexes. While this is not surprising, it is nevertheless somewhat disappointing. Clearly, a sound quality metric cannot be based mainly on subjective quantifiers like programmer ratings. The failure of the Halstead metrics to outperform the ratings suggests that either the measures are not good candidates for quality metrics or that non-linear relationships may be more appropriate.

On the plus side, **QR**, **QE**, and **QM**, are all dependent on **L**, **T**, and **V** if we take into account the relationships of equations 3-4. The indication here is that these three may be explored further to ascertain their potential as parameters in quality metrics. Noticeably, all the coefficients for the programming time, T, are greater than zero. The implication is that the programs requiring more time and mental discriminations to complete are of better quality. It should be observed that all of the results apply to programs which implement the same set of requirements. Therefore, a model developed using this methodology would be useful for selecting the best product among a set of similar, reusable modules. For the interested reader, a detailed account of all the models and the statistical values obtained can be found in reference 13.

Summary

A pilot study was conducted to assess the feasibility of using Halstead measures to develop a software quality metric. A FORTRAN program was assigned to a group of students and a set of four quality indexes was computed from those programs. The measures along with a programmer rating were used as regressors in a distinct linear model for each index.

Although linear models were derived for three of the four indexes which included the Halstead measures, in each case the programmer rating showed a better correlation than any of the measures. The three models can be expressed in terms of the program volume, the program level, and the programming time. Thus, **L**, **V**, and **T** could be further researched to establish more solid linear relation hips or explore non-linear versions.

References

[1] B. Beizer, "Software System Testing and Quality Assurance," Van

Nostrand Reinhold, New York, 1984.

[2] P. C. Belford and R. A. Berg, "Central Flow Control Software Development: A Case Study of the Effectiveness of Software Engineering Techniques," Proceedings of the Fourth International Conference on Software Engineering, Munich, September 1979.

[3] B. Curtis, et al., "Third Time Charm: Stronger Predictions of Programmer Performance by Software Complexity Metrics," Proceedings of the Fourth International Conference on Software Engineering, Munich, 1979.

[4] B. Curtis, "The Measurement of Software Quality and Complexity," Software Metrics, ed. by A. Perlis, et al., MIT Press, Cambridge, 1981.

[5] C. Daniel and F. Wood, "Fitting Equations to Data," John Wiley & Sons, New York, 1980.

[6] J. S. Davis and R. J. LeBlanc, "A Study of the Applicability of Complexity Measures," IEEE Transactions on Software Engineering, Volume SE-14, Number 9, September 1988, pp. 1366-1371.

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

[8] D. Kafura and G. R. Reddy, "The Use of Software Complexity Metrics in Software Maintenance," IEEE Transactions on Software Engineering, Volume SE-13, Number 3, March 1987, pp. 335-343.

[9] M. Lipow and T. A. Thayer, "Prediction of Software Failures," Proceedings to the Annual Symposium on Reliability and Maintainability, January 18-20, 1977.

[10] C. L. Mallows, "Some Comments on Cp," Technometrics, Number 15, 1973.

[11] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Volume SE-2, Number 4, December 1976, pp. 308-320.

[12] L. M. Ottenstein, "Quantitative Estimates of Debugging Requirements," IEEE Transactions on Software Engineering, Volume SE-5, Number 5, September 1979, pp. 504-514.

[13] A. R. Puerta, "A Linear Regression Approach to Develop a Software Quality Metric Based on Halstead Measures," M.S. Thesis, Tennessee Technological University, 1987.

[14] N. F. Schneidewind, "Software Metrics for Aiding Program Development Debugging," Proceedings of the 1979 National Computer Conference, Montvale, New Jersey, 1979.

[15] N. F. Schneidewind and H. M. Hoffman, "An Experiment in Software Error Data Collection and Analysis," IEEE Transactions on Software Engineering, Volume SE-5, Number 3, May 1979, pp. 276-286.

[16] V. Y. Shen, et al., "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support," IEEE Transactions on Software Engineering, Volume SE-9, Number 2, March 1983, pp. 155-165;

[17] J. M. Stroud, "The Fine Structure of Psychological Time," Annals of New York Academy of Sciences, pp. 623-631, 1966.