A Maintainability Model for Industrial Software Systems Using Design Level Metrics

S. Muthanna[1], K. Kontogiannis[2], K. Ponnambalam[1], B. Stacey[3]

University of Waterloo
Dept. of Systems Design Engineering,[1]
Dept. of Electrical & Computer Engineering[2]
Waterloo ON. N2L 3G1
Canada

Nortel[3]
SEAL Lab
Ottawa
Canada

Abstract

Software maintenance is a time consuming and expensive phase of a software product's life-cycle. This paper investigates the use of software design metrics to statistically estimate the maintainability of large software systems, and to identify error prone modules. A methodology for assessing, evaluating and, selecting software metrics for predicting software maintainability is presented. In addition, a linear prediction model based on a minimal set of design level software metrics is proposed. The model is evaluated by applying it to industrial software systems.

Keywords: *Software maintenance, software metrics, software quality.*

1 Introduction

According to studies presented in the literature [Schach96], [Sommer96], software maintenance accounts for more than sixty percent of the costs in the software life cycle. In many cases it is crucial to be able to identify the components of a system that may exhibit low maintainability and low software quality [Ponna96]. In the literature [Sneed85] a number of features have been associated with maintainability and software quality [Ponna96]. These include:

- *Impact rate*, defined as the percent change required in the system to perform a maintenance task (corrective, adaptive, perfective)
- *Effort*, defined as the time spent for performing a maintenance task
- *Error rate*, defined as the percent change in the number of errors introduced after a maintenance task,
- *Subjective evaluation*, defined as a weighted sum of subjective values provided by the software engineers involved with the system. The values often represent key design and implementation characteristics of the system such as cohesion, coupling, complexity, and documentation.

The problem of predicting the maintainability of a software system has been discussed extensively in the literature [Khosh99a], [Khosh99b], [Oman94], [Arnold93], [Peercy81]. In most cases the estimate is based on a model that is built by using measurements of the features above. An analysis on how software quality and software metrics are related is presented in [Ponna96]. The approach discussed in this paper is based on the statistical analysis of

*This work was funded by the Nortel - Seal Lab Ottawa, The Natural Sciences and Engineering Research Council of Canada, and the Consortium for Software Engineering Research. The authors would like also to thank IBM Canada Ltd., Centre for Advanced Studies for its technical support. Inquiries for this paper can be sent to kostas@swen.uwaterloo.ca or ponnu@vlsi.uwaterloo.ca

different software metrics and the corresponding subjective maintainability estimate values, provided by programmers with in-depth knowledge of the subject systems. The work presented in this paper focuses on design level metrics that characterize the overall data and control flow between modules in contrast to the code level metrics (i.e Halstead type of metrics) discussed in other approaches [Oman94], [Ganes99].

Our approach consists of two major phases. The first phase is to assess and select a minimal set of metrics which exhibit desired properties such as, low correlation with one another, do not depend on any subjective estimate provided by the maintainers, and relate to different design properties of the system. The second phase focuses on the construction of a linear model using polynomial regression. The model has been developed by analyzing applications in a wide spectrum of domains (compilers, real time speech recognition systems, rule based pattern matching and inference systems). However, other maintainability prediction models specific to an application domain (i.e telephone switching) can also be developed by following the methodology discussed in this paper. This paper is organized as follows; Section 2 discusses related work, section 3 discusses a methodology for selecting, assessing and, developing the prediction model. In section 4 we present experimental results and we validate the model. In section 5 we discuss the limitations of the model and, in section 6 we discuss possible future work.

2 Related work

This section discusses a number of different models to measure maintainability that have been proposed in the literature. In [Berns84] a Maintainability Analysis Tool (MAT) using FLECS (a structured Fortran preprocessor) for the purpose of analyzing programs written in VAX-11 Fortran (superset of Fortran 77) was proposed. MAT operates at the syntactic level. A software component is analyzed in tokens and structures. Each structure can be associated with a maintainability index. By aggregating the partial maintainability indexes, MAT computes an overall maintainability value.

In [Sneed85] a software quality assessment environment SOFTING is proposed. SOFTING provides assessments on eight design attributes, namely: modularity, generality, portability, redundancy, integrity, complexity, time and span-utilization. These design attributes are measured by a number of different metrics. However, in many cases it may be very difficult to quantify all these attributes for a large industrial system.

A partial solution to the problems encountered in [Sneed85], is presented in [Oman92], [Oman94] where a maintainability assessment taxonomy to demonstrate the hierarchical nature of maintainability attributes is proposed. The taxonomy first divides maintainability into three factors: the management, the operational environment and the target software system. Each of these three factors are then successively decomposed until a set of simple attributes are identified and defined via metrics. In [McCall77], a model to measure maintainability based on a combination of quality criteria such as simplicity, conciseness, self-descriptiveness, and modularity is presented. In ISO 9126 [Kitchenham90] four quality criteria are recommended for measuring maintainability. These include, analyzability, changeability, stability, and testability. In [Peercy81] a maintainability prediction model is proposed based on six software attributes namely, modularity, descriptiveness, consistency, simplicity, expandability, and, instrumentation. In [Arnold93] a model for evaluating the usefulness of re-engineering from the maintainability point of view is proposed. Finally, in [Oman94] a maintainability prediction model is proposed based on Halstead's metrics. This approach is based on polynomial regression analysis. In this respect, the process is similar to the approach presented in this paper. However, it differs on the selection criteria of the software metrics used to develop the prediction model. In [Oman94] the authors focus on code level metrics, while we focus on module-level design related metrics. Thus we aim on predicting maintainability on a higher level of granularity, by examining the interface between modules, global data flow, and control flow complexity.

Within this context a number of techniques for developing maintainability prediction models have been presented in the literature. These techniques include:

- *Hierarchical multidimensional assessment.* In this method, hierarchical structure of attributes of the source code is developed for the purpose of measuring software maintainability. The dimensions considered to construct the hierarchical structure are: control structure, information structure, typography, naming conventions and, commenting. These three dimensions constitute the highest level in the hierarchy of source code maintainability. Metrics defined for each dimension of the hierarchical structure, can be further combined to calculate maintainability up the hierarchy.

- *Aggregate complexity measure using entropy.* In this method, a software system can be divided into a set of k functional modules where complexity for each module can be measured by n metrics $M_1, ..., M_n$. The informational measure of dependence between $M_1, ..., M_n$, is directly related to the complexity of the system and can be approximated [VanEmden71] by using the entropy value of the i^{th} metric, $i = 1, ...n$.

- *Principal components analysis.* This statistical technique is used to reduce collinearity between indepen-

dent variables and reduce the number of components used to construct regression models. The method for measuring software maintainability by constructing regression models will be discussed in the following section. Principal component analysis orthogonalizes the metrics into the new components, called principal components. The components having the most information are then selected to construct a maintainability regression model.

- *Factor analysis.* Given a set of n metrics, $M = (M_1, ..., M_n)$, with a mean value and a covariance matrix, this statistical technique groups the metrics by explaining the covariance structure in such a way that the metrics within the particular group are highly correlated, but with relatively small correlations to metrics in different groups. Each group of metrics represents a single underlying factor. Using this technique a complexity model for a set of n software components can be derived [Munson90].

- *Polynomial regression.* Regression analysis is a statistical method for predicting values of one or more dependent variables from a collection of independent variables. In order to assess software maintainability, one approach is to construct a polynomial equation where the maintainability of a system is expressed as a function of the associated metric attributes [Oman94] [Khosh92]. The following section discusses the process for building a maintainability model using this type of polynomial regression and software metrics that can be derived from the overall design properties of the system.

3 Development of the Prediction Model

In this section, we discuss the process of developing a maintainability prediction model using software design metrics by applying polynomial regression techniques. The model presented here has very reasonable constraints (discussed in detail in Section 5.), and thus it covers a wide range of software systems. However, individual organizations may require a modification of the proposed model to fit the characteristics of the software system at hand. The rest of this section is intended to provide a structured way for repeating the model generation process, and fine tuning it if necessary. In this paper we focus on the use of design metrics, but the same process can be performed by collecting other type of software related metrics. The model development process presented here has five major steps. These steps are discussed in detail in the following sections.

3.1 Metrics Selection

It has been argued that one can analyze a software system from different viewpoints [Holt96] [Muller93]. The first step on the maintainability analysis using metrics is to identify a collection of metrics that reflect the characteristics of the viewpoint with respect to which the system is being analyzed and, discard metrics that provide redundant information (i.e. provide the same information as other metrics already in the selected set).

In this particular work we are interested on examining the maintainability of the system from the design point of view and therefore we focus on collecting design related metrics. In [Kitchenham90] design metrics were classified in two major categories: *a)* Product metrics derived from design representations (e.g. Fan-in, Fan- out, cyclomatic complexity); *b)* Process metrics derived from the activities and tasks that make up the design process (e.g. effort and time-scale metrics, fault and change metrics) [Khosh98]. Some of the product design metrics that are commonly used in practice [Card88] [Yu93], [Schnei94] [Adamov87] are listed below. These metrics formed the first rough collection of metrics to be considered for the prediction model. A detailed description of these metrics can be found in [Adamov87]. These metrics include:

1. Albrecht metric (Function point metric)
2. Benyon-Tinker's software complexity metric
3. Card and Agresti's system complexity metric
4. Chapin's Q metric
5. Data complexity metric (D-complexity)
6. Data flow
7. Henry and Kafura information flow metric (Kafura)
8. Knot count
9. Local complexity
10. McCabe's (extended) cyclomatic complexity metric
11. McClure's program complexity metric
12. Nesting level
13. Relative complexity metric
14. Shepperd's IF4 metric
15. Span of control
16. Structural complexity metric (Fan-out)
17. Yau and Collofello's design stability metric
18. Yin and Winchester's network metric

The second goal of this process step is to identify a minimal set of design metrics that are relatively easy to obtain and, are useful in predicting software maintainability. With this objective in mind we identified metrics which are hard to compute (i.e. 'Yau and Collofello's design stability metric', 'Yin and Winchester's network metric') or contain redundant information with respect to the other (i.e use common software features). For example, 'Card and Agresti's

system complexity metric' 'Henry and Kafura information flow metric' 'McClure's program complexity metric' 'Relative complexity metric' and 'Shepperd's IF4 metric' all use the 'fan-in' and 'fan-out' concepts. This selection can be automated by applying a Spearman-Pearson correlation test to identify highly correlated metrics from the initial set. Highly correlated metrics were flagged and discarded. The metrics were computed for 32 modules (files) of ANSI C programs, ranging in size from 500 to 2000 lines of code with total size of approximately 55KLOC. This data set was used to develop the maintainability model. A separate data set of approximately 92KLOC was used to validate the model. Following this selection process. the metrics that have been considered from the initial set include:

1. Module level Function Point Metric
2. Module level Information Flow Metric
3. Module level Global Data Flow
4. Average Structural Complexity Metric (Fan-out)
5. Average Knot Count
6. Average Cyclomatic Complexity Metric

3.2 Data collection

The second step aims on the collection of subjective maintainability data from the software developers. This collection of data will form the core against which the metric values will be compared to develop the prediction model. The questionnaire used to gather the subjective data was adapted from [Oman94]. In [Oman94] it has been mentioned that this questionnaire has also been used in similar studies. The questions require answers to be given on a five point scale (very poor, poor, medium, good, very good) and all questions must to be answered. The engineers in charge of maintaining the design and code completed the surveys. The purpose of the five point responses was to provide a numerical rating that could be used as a dependent variable for polynomial regression analysis. A summation of the 32 responses was calculated by tallying the point values for each response, with values ranging from 1 for "very poor" to 5 for "very good". Software with a maintainability index $<$ 65 is considered "low" maintainability, between 65 and 85 is considered "medium," and software with a maintainability index $>$ 85 is considered to have "high" maintainability [Coleman95].

3.3 Correlation Analysis

The third step involves the selection of the metrics which are the strongest maintainability predictors. This can be

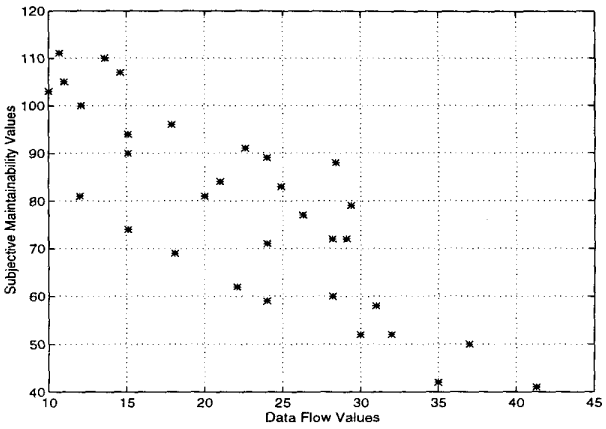

Figure 1. Subjective Value vs. Data Flow.

accomplished by examining the correlation between each metric and the subjective ratings provided by the software developers. The metrics can be ranked and the development of several models is possible. For example if the best two metrics are chosen a two metric model is possible, while if the three best metrics are selected a three metric model can be developed, and so on.

In our experiment, the total and average values per module (i.e.total value/ number of functions) of these five metrics were analyzed by the Pearson correlation test against the subjective values. A list of the metrics ordered by the correlation coefficients was obtained. Metrics with high correlation coefficients are potential candidates in predicting maintainability. The correlation relationships (R value) between the subjective value and the total metric variables have been computed as fan-out = -0.547; McCabe = -0.536; Albrecht = -0.197; Kafura = 0.199; total data flow = -0.864. Note that, these metrics are not as strongly correlated among themselves (because of the pre-selection at step 1). The corresponding correlation coefficients using average metric values and subjective values were computed as : average fan-out = -0.880; average McCabe = -0.728; average Albrecht = -0.264; average Kafura = 0.043; and average data flow = -0.359. The negative coefficients indicate that as the complexity of the software increases as measured by these metrics, the ease of maintenance goes down. Graph plots lend support to the conclusion we drew on the basis of the correlation coefficient table alone. The scatter-plots allow us to see the relationships such as a curvilinear pattern between two quantitative variables that descriptive statistics do not reveal. An example consider the scatter plot for the Data Flow metric is illustrated in Fig.1, which shows a possible "linear" pattern indicating high correlation values.

3.4 Regression Analysis and Model Development

The objective of the fourth step is first to determine a single maintainability prediction value by combining a minimal set of metrics. We call this maintainability prediction value the *Maintainability Index*, which when developers provide is the subjective rating. The second objective is to determine the best possible regression model for the data being considered. Correlation and regression are equally important, and a complete analysis of the relationship between two variables includes both.

In this step a series of linear regression tests were carried out using the selected metrics (avg. fan-out, data flow, avg. McCabe, fan-out and McCabe) as independent variables for predicting the subjective maintainability value provided by the developers. The regression tests aimed to determine:

- How these metrics could be combined into a single maintainability index
- What is the best possible regression model for the data being considered

Once alternative models were constructed, the combination of the selected metric variables and their coefficients for the best model were chosen based on:

- *goodness-of-fit* statistic tests. These tests include the correlation coefficients of the suggested model with the subjective values. In particular, the values of R, R^2, adjusted R^2, and standard error of the estimate were considered. All metric variables in the model must also pass the tolerance criterion which for this experiment was set to 0.0001. A metric variable also is not considered in the final model if it causes the tolerance of another variable to drop below the pre-set tolerance criterion. The regression analysis indicated that fan-out and McCabe metrics were not significant contributors for building a regression model compared to that of average fan- out, data flow and average McCabe metrics.
- *analysis-of-variance* tests. These tests used the F-statistic to test the hypothesis that the estimates of the true errors in the model follow a normal distribution (the points fall in an horizontal band with no apparent systematic features[1]). In this statistic test F is large when the independent variables help explain the variation of the dependent variable. In our experiments the linear relation is highly significant (F = 59.17 with significance $\leq$ 0.0005)
- *analysis-of-regression-coefficients* test. The t statistics values provide some insight regarding the relative importance of each variable in the model. The value of t is obtained by dividing each coefficient by its standard error [SPSS96]. As a rule of thumb, t values below -2 or above +2 give an idea regarding useful predictors and indicate that they are not correlated among themselves. Hence in our analysis, the t values, -3.515, -3.390, and -2.388 for average fan-out, data flow, and average McCabe respectively meet the guideline.

As our objective was to develop a model using minimal set of metrics with high R^2 (R-square) value, we restricted ourselves to average fan-out, data flow and average McCabe metrics for building a regression model.

R is the Spearman correlation factor between the observed and predicted values of the dependent variable. R^2 indicates the amount of the dependent variable information that can be obtained by using the predicted variables in the maintainability model. In our experiments the combination of the metrics above metrics give an R^2 value of 0.85. Regression coefficients were computed using standard polynomial regression techniques [SPSS96]. Given the above parameters, our estimated regression three-metric linear model has been estimated as:

$$SMI = 125 - 3.989 \cdot FAN_{avg} - 0.954 \cdot DF - 1.123 \cdot MC_{avg}$$

where,

- SMI is the Software Maintainability Index (predicted value with value ranges 0-125) for a given module,
- FAN_{avg} is the average number of external calls emanating from the module,
- DF is the total number of outgoing and incoming data flow for the module and,
- MC_{avg} is the average McCabe for the module.

4 Model Validation

This section discusses the validation of the estimated model. As a matter of model's validity it must be verified that:

- the data does not contain values that violate any assumptions considered for the development of the model. This can be achieved by plotting standardized residual values against standardized predicted value of the dependent variable. If the model is appropriate for the data, the residuals should follow a normal distribution (the points fall in a horizontal band with no apparent systematic features).

[1] see also Fig.3

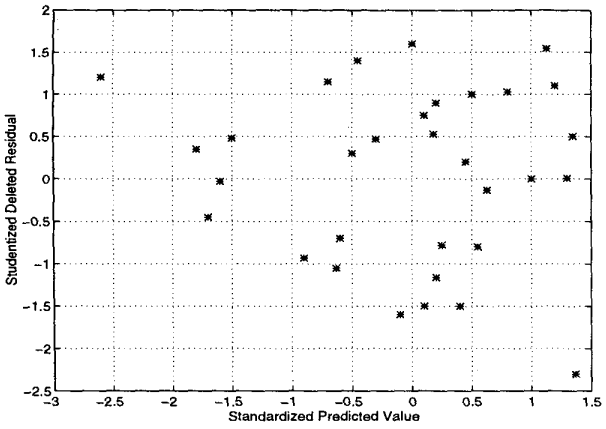

Figure 2. Standardized Predicted Value vs. Studentized Residual Value

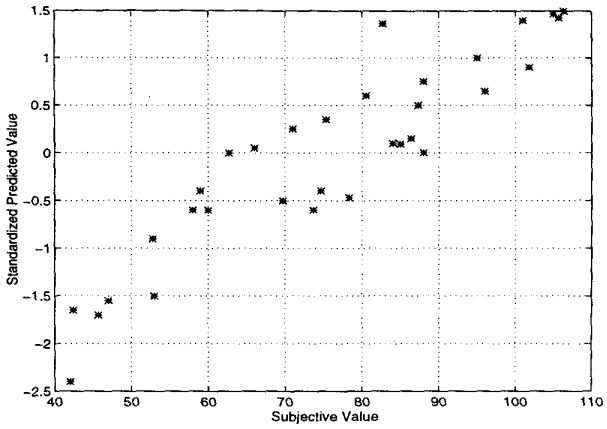

Figure 3. Subjective Value vs. Standardized Predicted Value.

- regression assumptions are satisfied. This can be achieved by plotting the Studentized deleted residuals against standardized predicted values. This plot is illustrated in Fig.2. Many researchers prefer using this plot, because, when the model used to compute the residuals is valid, they have mean 0 and variance 1. If the errors are normally distributed and the form of the model is correct, then about 95% of the residuals should fall between -2 and +2 values. For each module, a deleted residual can be calculated by excluding the module from the calculation of the regression coefficients (this procedure was repeated for each module in turn).

Another validation plot is the standardized predicted values vs. observed (subjective) values to check the data coincidence. If the model fit each data value exactly, the observed and predicted values would coincide on a straight line extending from the lower left corner to the upper right (linear y = x type of relation). This plot is illustrated in Fig.3. To test the applicability of the polynomial model, a separate set of ANSI C programs data must be tested against the model. The subjective values must be obtained by using the same questionnaire that has been used to collect the data for building the model.

In our validation experiment 92KLOC of C source code, taken from large and medium sized industrial systems were considered. The metric values were used in the model for assessing the maintainability index. The observed values provided by the developers along with the proposed model's predicted values for a subset of our validation data are given in Table.1. The difference between the predicted value and the observed (subjective) value of our validation data are close enough in most of the cases.

Here, scatter-plots of the subjective value versus the model's predicted value provides useful validation information as well. The validation plots and table (Fig.2, Fig.3, Table.1), provide evidence that the model is fit for the test modules. We may statistically infer that the maintainability assessment polynomials presented here are statistically accurate models of the test data on which they were constructed. A detailed set of validation tests can be found in [Muthann97].

5 Model Restrictions

This model can be used to measure the maintenance difficulties of systems that can be broken down to modules (source files) of medium size (i.e. 1KLOC - 2KLOC source lines per module (file). The working ranges of values for this model are between 0-14 for the average values of fan-out and McCabe metrics and 0-45 for data flow metric. We believe that these are very reasonable restrictions (one does not expect average McCabe more than 14, or more than 14 function calls per function, or more than 45 global variables updated/used within a single function). The objective is that this model would help software developers identify difficult to maintain modules before integrating all software modules together. If the complexity of the system is very high, having very high value of fan-out, data flow, and McCabe, the model's maintainability index becomes off-scale negative which again indicates poor level of maintainability. Although the model can be applied in a wide range of procedural software systems, we do not claim that this is the only model for predicting maintainability, nor that is the

best overall. However, it is a model that can be automated and used to quickly and easily predict software maintainability. New models tailored to particular application domains can also be built following the process discussed in this paper.

6 Conclusion

Software maintainability is going to be a continuing challenge for many years to come. It is believed that predicting the maintenance at the design level will help software designers and maintainers to alter the architecture of the software system for better performance that leads to the overall reduction of maintenance costs. The intent of this paper is to develop an automated working maintainability model using design level metrics. At the module level this model can be used to monitor changes to the system as they occur, and as a method of predicting error prone modules. The proposed model *a)* focuses on the design aspects of the software; *b)* does not depend on subjective metrics; *c)* is fast to compute and, *d)* is robust (linear). Although this model may not be perfect in all environments, it demonstrates the utility of developing such models tailored for particular application domains. Developers and maintainers can follow the process described in this paper to further customize the maintainability model for their application domain, if needed. The point is that a good model at an early stage (design) of the software life cycle can help maintainers guide their efforts and provide them with much needed feedback.

The applications presented in this paper are only a few of the uses for maintainability prediction. There are many more directions this work could take in the future. Possible topics for further research that will provide valuable information about industrial software development include: *a)* the use of automated maintainability prediction models as acceptance criteria for software quality control, third party software procurement and, in-house product maintenance (i.e system restructuring to achieve maintainability criteria); *b)* the analysis on how maintainability differs between functional and object oriented systems and finally, *c)* the analysis of how software maintainability affects other software quality factors such as reliability, re-usability and portability.

Acknowledgment

The authors would like to thank Chris Tandy and Cinderella Lee of IBM Canada Ltd., for providing assistance and valuable insights for the development of the proposed model. We also thank the Consortium for Software Engineering Research members for their useful comments on various CSER technical meetings where preliminary versions of this research were presented.

References

[Adamov87] Adamov, R. "Literature review on software metrics", *Zurich: Institut fur Informatik der Universitat Zurich*, 1987.

[Arnold93] Arnold, R., "Software Reengineering", IEEE Computer Press, 1993.

[Berns84] Berns, G., "Assessing Software Maintainability", *Communications of the ACM*, vol.27, no.1, pp.14-23.

[Coleman95] Coleman, D., Lowther, B., Oman, P., "The Application of Software Maintainability Models in Industrial Software Systems", Journal of Systems Software, vol.29, pp.3-16.

[Card88] Card, D., Argesti, W., "Measuring Software Design Complexity", *Journal of Systems and Software*, vol.8, pp.185-197.

[Ganes99] Ganesan, K., Khoshgoftaar, T., Allen, E., "Case-based software quality prediction", *International Journal of Software Engineering and Knowledge Engineering, 9(6)*, 1999.

[Holt96] Holt, R., Pak., J., "GASE: Visualizing Software-Evolution-in-the-Large" *Working Conference on Reverse Engineering*, Monterey, CA., 1996, pp.163-167.

[Khosh92] Khoshgoftaar, T.M., Bhattacharya, B.B., Richardson, G.D., "Predicting Software Errors, During Development, Using Nonlinear Regression Models: A Comparative Study", *IEEE Transactions on Reliability, 41:3*, 1992, pp. 390-395.

[Khosh98] Khoshgoftaar, T., Allen, E., Halstead, R., Trio, G., Flass, R., "Using Process History to Predict Software Quality", *IEEE Computer 31(4)*, 1998, pp. 66-72.

[Khosh99a] Khoshgoftaar, T., et.al., "Which software modules have faults that will be discovered by customers?" *Journal of Software Maintenance: Research and Practice, 11(1)*, January 1999, pp.1-18.

[Khosh99b] Khoshgoftaar, T., et.al., "Predicting fault-prone software modules in embedded systems with classification trees", *In Proceedings: Fourth IEEE International Symposium on High-Assurance Systems Engineering*, Washington, DC USA, November 1999. IEEE Computer Society, pp. 105-112.

[Kitchenham90] Kitchenham, B., Pickard, L., Linkman, S., "An Evaluation of Some Design Metrics", *Software Engineering Journal*, vol.5, no.1, pp.50-58.

[McCall77] McCall, J., Richards, P., Walters, G., "Factors in Software Quality", *National Technical Information Service*, vol.1,2 and,3, AD/A-049-014/015/055.

[Munson90] Munson J., Khoshgoftaar, T., "Applications of a Reactive Complexity Metric for Software Project Management", *Journal of Systems Software*, vol.15, 1990, pp.283-291.

[Muller93] Muller, H., Orgun, M., Tilley, S., Uhl, J., "A Reverse-engineering Approach to Subsystem Structure Identification", *Journal Software Maintenance Research and Practice*, vol.5., pp.181-204.

[Muthann97] Muthanna, S., "Assessing Maintainability of Industrial Software Systems Using Design level Metrics", *M.Sc. Thesis, Department of Systems Design Engineering*, University of Waterloo, Waterloo, ON., 1997.

[Ponna96] Ponnambalam, K., Kalaichelvan, S., Goel, N., Munikoti, R., "Analyzing Sensitivities of Software Qualities to Various Metrics", *The Sixth International Conference on Software Quality (6ICSQ)*, Ottawa, pp.346-355.

[Schach96] Schach S., "Software Engineering", Irwin Publishers, 1996.

[Sneed85] Sneed, H., Merey, A., "Automated Software Quality Assurance", *IEEE Transactions on Software Engineering*, vol.11, no.9., pp.909-916.

[Oman94] Oman, P., Hagemeister, J., "Construction of Testing Polynomials Predicting Software Maintainability", *Journal of Systems Software*, vol. 27, pp.251-266.

[Oman92] Oman, P., Hagemeister, J., Ash, D., "A Definition and Taxonomy for Software Maintainability Assessment", *Software Engineering Test Lab Technical Report*, University of Idaho, 1992

[Peercy81] Peercy, D., "A Software Maintainability Evaluation Methodology", *IEEE Transactions on Software Engineering*, vol.7, July 1981, pp.343-352.

[Schnei94] Schneidewind, N., "Validating Metrics for Ensuring Space Shuttle Flight Software Quality", *IEEE Computer 27(8): 50-57*, 1994.

[Sommer96] Sommerville, I., "Software Engineering", Addison Wesley, 1996.

[SPSS96] "SPSS Base 7.0 for Windows User's Guide", SPSS Inc.

[VanEmden71] VanEmden, M., "An Analysis of Complexity", *Mathematical Centre Tracts*, Amsterdam.

[Yu93] Yu, X., Lamb, D., "Metrics Applicable to Software Design", *Software Quality Management*, vol.10, 1993.

Module Id	Subjective Index	Subjective Rating	Model Index	Model Rating
1	42	Low	45.71	Low
2	46	Low	44.59	Low
3	41	Low	33.74	Low
4	57	Low	66	Low
5	60	Low	65.89	Low
6	58	Low	70.54	Medium
7	53	Low	50.30	Low
8	86	Medium	79.71	Medium
9	88	Medium	89.28	High
10	84	Medium	78.47	Medium
11	81	Medium	102	High
12	72	Medium	92	High
13	74	Medium	70.97	Medium
14	74	Medium	70.97	Medium
15	95	High	95.13	Low
16	100	High	99.85	High
17	101	High	115	High
18	104	High	117	High
19	93	High	77	Medium
20	89	High	102	High
21	81	High	88.39	High

Table 1. Results of Model Application