Maintainability Prediction: A Regression Analysis of Measures of Evolving Systems

Jane Huffman Hayes
Computer Science Department
University of Kentucky
hayes@cs.uky.edu
(corresponding author)

Liming Zhao
Computer Science Department
University of Kentucky
lzhao2@uky.edu

Abstract

In order to build predictors of the maintainability of evolving software, we first need a means for measuring maintainability as well as a training set of software modules for which the actual maintainability is known. This paper describes our success at building such a predictor. Numerous candidate measures for maintainability were examined, including a new compound measure. Two datasets were evaluated and used to build a maintainability predictor. The resulting model, Maintainability Prediction Model (MainPredMo), was validated against three held-out datasets. We found that the model possesses predictive accuracy of 83% (accurately predicts the maintainability of 83% of the modules). A variant of MainPredMo, also with accuracy of 83%, is offered for interested researchers.

1. Introduction

As society becomes more and more dependent on software and demands that new, more capable software be provided on short cycles, the need for maintainable, reliable software continues to increase. In this paper, we are interested in improving the maintainability of such evolving software systems. Maintainability is the ease with which software can be corrected, errors, adapted to environment changes, and/or enhanced per customer requests[8]Our suggested approach for improving the maintainability of software is to first understand the characteristics of maintainable software and attempt to measure these characteristics in order to determine the maintainability of a product .

We performed correlation analysis and found that coding effort correlates with maintainability. We developed a new measure that captures the relationship between requirements and design effort and coding effort. Next, we built a regression model, Maintainability Prediction Model (MainPredMo), using the new measure and two datasets. We then used the model to categorize three datasets of software modules, where each module was designated as "easy to maintain" or "not easy to maintain." The model was found to have 83% accuracy.

2. Related Work

There is no clear agreement on how to measure maintainability. Welker and Oman suggest measuring it statically by using a Maintainability Index (MI) which is decided by cyclomatic complexity, lines of code (LOC), and lines of comments [8]. Ramil and Lehman suggested a linear model with effort and %modules changed as the variables.[7] Polo, Piattini, and Ruiz used number of modification requests, mean effort per modification request, and type of correction to examine maintainability [5]. Pressman [6] also introduced an effort-based metric, mean-time-to-change (MTTC), to predict maintainability. A lengthier survey of related work is presented in [2] and [3].

3. Maintainability Prediction Model (MainPredMo)

To build the predictive model, we first identified static measures of interest. We performed correlation analysis on these measures. We then used regression analysis on the datasets to build the model. Finally, we validated the model. Each will be discussed below, as well as threats to validity. We hypothesize that the maintainability of a software application can be measured using static metrics that can be derived from the software development process. Specifically, we hypothesize that coding effort, percentage of modules changed, design effort, MI, classes changed, classes added, comment ratio (CR), Attribute Complexity (AC) [1], True comment ratio (TCR), lines of code (LOC), and requirement effort correlate well with maintainability. Further, we hypothesize that a model can be built to predict the maintainability of software, given these static metrics.

To examine these hypotheses, we first focused on the static metrics that might affect the effort required to maintain a software program. Hayes et al present a subjective measure, "perceived maintainability" (PM). The maintaining software engineer, after all changes have been made, assign a value from 1 to 10 to each component modified, where 10 is code that was very easy to change [2, 3]. We treated perceived maintainability (PM) as the dependent variable (typically denoted as y) and the hypothesized metrics as independent variables. Since our first step was to determine whether a relationship existed between these two quantitative variables, we performed correlation analysis (Section 3.3) for each of the metrics presumed related to effort. Once correlating metrics were identified, we performed regression analysis to derive an analytical model as discussed in Section 3.4.

3.1 Data Sets

We used data from five sources. Three data sets were from software engineering (or related) classes, including CS499 of Fall 2001, CS616 of Fall 2002, and CS616 of Fall 2004; Spathic Project Data was from the source code a test generation tool; PA0 and PA2 were from the student projects of two networking classes. We expected to get adaptable results (that could be easily generalized) due to the heterogeneity of the data. Parts of the data are based on object-oriented design and programming, so our experiment results apply to OO projects as well.

3.2 Threats to Validity

We attempted to limit internal validity threats by validating the tools and processes we used for data collection and analysis. A major threat to external validity (generalization of results) for our work is the representativeness of our subject programs and changes as well as our small sample size. Also, we worked with students and with student applications. However, Høst et al [4] found that students perform the same as professionals on small tasks of judgment.

3.3 Correlation Analysis

We used PM as the dependent variable, performing correlation analysis with the candidate measures listed in Section 3. Correlation analysis infers whether a linear relationship exists between two variables. That is to say, the higher the value of the coefficient of determination, the stronger the relationship between actual maintainability (PM) and the metric. We ranked the candidate measures by their determination value. Coding effort for change is ranked first with the highest value of determination value or R^2 (0.82) and a very low significance value (.0003). The high correlation value (very close to 1) indicates that this metric increases positively as perceived maintainability increases, in almost perfect correlation. The significance value indicates that there is less than a 1% chance (.0003) that these results are due to chance. %Modules changed is ranked second with the determination value of 0.26 and significance value of 0.13. None of the other measures show very impressive determination values or significance values. The fact that lines of code (LOC) is not a crucial factor in our experiment might be a little surprising, at first glance. However, if we consider the following factors, the result is reasonable: LOC is just one of the factors that influences maintainability. Other important factors include complexity, code annotation, documentation, etc. Lines of code alone cannot determine the maintainability of the code.

Before proceeding, we analyzed our environment by performing some basic prescriptive analysis on the data sets. Table 1 indicates that our models were built based on small- to medium-sized maintenance tasks. Note that Total effort is the sum of requirement effort, design effort, and coding effort, measured in person minutes.

3.4 Development and Validation of Maintainability Prediction Model

"A typical estimation model is derived using regression analysis on data collected from past software projects [5]." The heterogeneity of the data is one of the strengths of our study. For example, one project was a stereology application (image processing) and one was a nutrition tracker. Heterogeneous data helped us achieve more generalized results than data from just one particular domain. The primary description of this data is found in Section 3.1. However, this section provides some additional details.

Table 1. Descriptive statistics for LOC and Total Effort.

LOC		**Total effort (minutes)**	
Mean	2211.56	Mean	117.5
Minimum	701	Minimum	25
Maximum	9542	Maximum	255

3.4.1 Compound Metric – RDCRatio

In the above process, we found that Coding effort ("effort" for the change) affects the maintenance effort and hence maintainability. Considering that coding is not the only activity involved in maintaining software and/or in adding functionality to evolving software, we need to analyze how coding effort compares with other elements of the software process, such as design effort and requirement analysis effort.

We decided to bring in other metrics as mentioned in the previous section. Since requirement effort and design effort are the most relevant items, we attempted to build a compound metric out of the three (the Requirement/Design/Code ratio or RDCRatio) in order to normalize the code effort data.

RDCRatio = (Reqt Effort + Design Effort)/Coding Effort

There are two possible approaches to constructing the prediction model using RDCRatio: using regression analysis to construct a prediction model, or using the related rank of the RDC ratio to predict maintainability directly. Next, we will show how we use the above two approaches.

3.4.2 Regression Analysis

By using the CS499 and CS616 project data to build a model, we obtained the following:

MainPredMo =3.795 + 1.652RDCRatio (1)

We used a total of 10 points to build the model shown in formula (1) and in Figure 1. Analysis showed that this model has a very low statistical significance value of 0.005 and an R square value of 0.64.

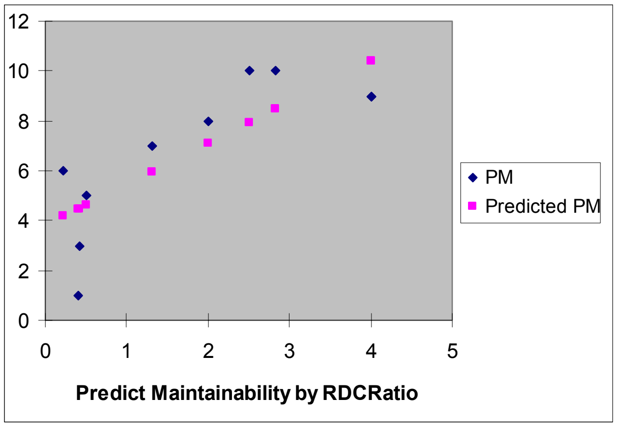

Figure 1. Predict maintainability using RDCRatio.

3.4.3 Using RDCRatio to Predict Maintainability Directly

We use RDCRatio rank instead of regression analysis to predict the perceived maintainability on two smaller data sets, PA0 and PA2. We ranked the projects by their RDCRatios and used that to predict the related rank of maintainability to identify the codes/modules that require re-factoring and optimization. To validate this notion, we also ranked the data sets by the PM values provided by the programmers, as shown in Table 2. If the manager makes the decision that half of all modules having the lowest RDCRatio need to be re-factored or modified to make them more maintainable, the modules chosen will be B, C, and F. As shown in Table 3, the model correctly classifies 5 of the 6 modules (83%). The model also correctly identifies all modules that are hard to maintain. The only error made is that an easy to maintain module (module C) is incorrectly classified as hard to maintain. We'd prefer to make such errors (though not frequently), as opposed to "missing" a hard to maintain module, since it would not hurt to review a module already possessing high maintainability.

Table 2. Ranking Data Sets by RDCRatio and by PM.

Data set	PM	PM Rank	DCRatio	DCRatio Rank
A	10	1	1.14	3
B	7	6	0.26	5
C	10	1	1	4
D	9	4	2.6	1
E	10	1	2	2
F	8	5	0.18	6

Table 3. Classification of maintainability by RDCRatio and its validation.

Module	(perceived) Maintainable	Classified by RDC Ratio
A	Yes	Yes
B	No	No
C	Yes	No
D	Yes	Yes
E	Yes	Yes
F	No	No

The two approaches were validated on the CS616-2 data set and both produced good results. The regression approaches achieved a significance value of 0.026 and a high R square value of 0.85, showing that RDCRatio did have strong correlation and thus is a good predicator of maintainability. In the RDC ranking approach, the data point with the highest RDCRatio (1.5) has the highest PM value (9), and the data point with the lowest RDCRatio (0.25) also has the lowest PM value (7).

3.5.7 Multiple metrics model

In this model, we treated the percentage of modules changed (%moduleschanged) and RDCRatio as multiple variables (Xs) for predicting maintainability.

In building the next model, we used the data from the CS499 and CS616 data sets.

MainPredMo_new = 4.083 + 1.602RDCRatio - 0.01%modules changed (2)

The R square is 0.64 and the p value is 0.028, below the desired significance value of 0.05.

4. Conclusions and Future Work

LOC, documentation, complexity, coding style, etc. are all factors that influence the maintainability of software. However, these factors act together and not any one of them single-handedly "decide" the software's maintainability. Due to this fact, the normalized coding effort, RDCRatio, provides an easy way to predict the software maintainability indirectly. We also found that %Modules changed, also an indirect measure, could be used together with RDCRatio to predict software maintainability.

We proposed a method for predicting the maintainability of a software application based on a ratio of the requirement and design effort to the coding effort, called RDCRatio, and percentage of modules changed. We first used our prior research to generate a list of possible measures that correlate with maintainability (maintenance effort). We performed correlation analysis and ranked the above measures. We found that of the measures we hypothesized to correlate with maintainability, only coding effort and a ratio of requirement effort and design effort to coding effort did so. This confirms our intuition that effort correlates with maintainers' perception and that the RDC ratio further offers a normalized metric that can be used to predict maintainability with two alternative approaches. After several iterations, we proposed a final multiple regression model, Maintainability Prediction Model (MainPredMo).

A larger scale study with a variety of industry projects across diverse domains is required before any broad conclusions can be reached. We also plan to apply other statistical techniques, such as principal component analysis, to the data sets.

5. Acknowledgements

We thank the students of CS 650 (January 2001), CS 499 (August 2001), CS 616 (Fall 2003 and Fall 2004), Inies Raphael Michael Chemmannoor, Tina Gao, and Senthil Karthikeyan Sundaram for their contributions. We thank Dr. Ken Calvert for allowing us to use specifications and code from his CS 571 course. We thank Togethersoft for their donation of Together to our research program.

6. References

[1] Chidamber S, Kemerer C. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 1994; 20(6):476–493.

[2] Hayes, J. Huffman, Mohamed, N., Gao, T. The Observe-Mine-Adopt Model: An agile way to enhance software maintainability. *Journal of Software Maintenance and Evolution: Research and Practice,* Volume 15, Issue 5, Pages 297 – 323, October 2003.

[3] Hayes, J. Huffman, Patel, S., and Zhao, L. A metrics-based software maintenance effort model. Proceedings of the 8th European Conference on Software Maintenance and Reengineering, CSMR 2004, Finland, March 2004.

[4] Høst M, Regnell B, Wohlin C. Using students as subjects – A comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering* 2000; 5(3):210–214.

[5] Polo M, Piattini M, Ruiz F. Using code metrics to predict maintenance of legacy programs: a case study. Proceedings of the International Conference on Software Maintenance, ICSM 2001. IEEE Computer Society: Florence Italy, 2001;202–208.

[6] Ramil J, Lehman M. Metrics of software evolution as effort predictors – A case study. Proceedings International Conference on Software Maintenance, ICSM 2000. IEEE Computer Society: San Jose CA, 2000;163–172.

[7] Pressman, R.S., *Software Engineering A Practitioner's Approach,* McGraw-Hill, 2001.

[8] Welker, K.D. and Oman, P.W. Software Maintainability Metrics Models in Practice, *Journal of Defense Software Engineering*, Volume 8, Number 11, November/December 1995, 19-23.