International Journal of Computer Science and Network (IJCSN)
Volume 1, Issue 1, February 2012 www.ijcsn.org ISSN 2277-5420

Modified Cocomo Model For Maintenance cost Estimation of Real Time System Software

[1]Sugandha Chakraverti, [2]Prof Sheo Kumar, [3]Dr. S.C. Agarwal , [4]Ashish Kumar Chakraverti

[1]Department of CSE, MBU
Solan, HP-173229, India

[2]Department of CSE, MBU
Solan, HP-173229, India

[3]Department of Mathematics, KCCEC
Greater Noida UP-201308, India

[4]Department of CSE, BBDIT
Ghaziabad, UP-201002, India

Abstract: Accurate cost estimation of software projects is one of the most desired capabilities in software development Process. Accurate cost estimates not only help the customer make successful investments but also assist the software project manager in coming up with appropriate plans for the project and making reasonable decisions during the project execution. Although there have been reports that software maintenance accounts for the majority of the software total cost, the software estimation research has focused considerably on new development and much less on maintenance. Now if we talk about real time software system(RTSS) development cost estimation and maintenance cost estimation is not much differ from simple software but some critical factor are considered for RTSS development and maintenance like response time of software for input and processing time to give correct output. As like simple software maintenance cost estimation existing models (i.e. Modified COCOMO-II) can be used but after inclusion of some critical parameters related to RTSS.

A Hypothetical Expert input and an industry data set of eighty completed software maintenance projects were used to build the model for RTSS maintenance cost. The full model, which was derived through the Bayesian analysis, yields effort estimates within 30% of the actual 51% of the time,outperforming the original COCOMO II model when it was used to estimate theseprojects by 34%. Further performance improvement was obtained when calibrating the full model to each individual program, generating effort estimates within 30% of the actual 80% of the time.

Keywords:- *COCOMO-II, RTSS, Bayesian analysis, Efforts Estimation.*

1. Introduction

Software maintenance is an important activity in software engineering. Over the decades, software maintenance costs have been continually reported to account for a large majority of software costs [Zelkowitz 1979, Boehm 1981, McKee 1984, Boehm 1988, Erlikh 2000]. This fact is not surprising. On the one hand, software environments and requirements are constantly changing, which lead to new software system upgrades to keep pace with the changes. On the other hand, the economic benefits of software reuse have encouraged the software industry to reuse and enhance the existing systems rather than to build new ones [Boehm 1981, 1999]. Thus, it is crucial for project managers to estimate and manage the software maintenance costs effectively.

1.1The Problem

COCOMO is the most popular non-proprietary software estimation model in literature as well as in industry. The model was built using historical data and assumptions of software (*ab initio*) development projects. With a few exceptions, the model's properties (e.g., forms, cost factors and constants) are supposed to be applicable to estimating the cost of software maintenance. However, inherent differences exist between software

development and maintenance. For example, software maintenance depends on quality and complexity of the existing architecture, design, source code, and supporting documentation. The problem is that these differences make the model's properties less relevant in the software maintenance context, resulting in low estimation accuracies achieved. Unfortunately, there is a lack of empirical studies that evaluate and extend COCOMO or other models, in order to better estimate the cost of software maintenance.

1.2 A Solution

Instead of using the COCOMO model that was designed for new development, what if we build an extension that takes into account characteristics of software maintenance. Thus, my work is on *Improved COCOMO models that allow estimators to better determine the equivalent size of maintained software and estimate maintenance effort can improve the accuracy of effort estimation of software maintenance.*

The goal of this study is to investigate such models. I will test a number of hypotheses testing the accuracy of alternative models that predict the effort of software maintenance projects, the explanatory power of such cost drivers as execution time and memory constraints, and potential effects on quality attributes as better determinants of the effort involved in deleting code. Using the industry dataset that I have collected, I will also evaluate the differences in the effects of the COCOMO cost drivers on the project's effort between new development and maintenance projects. Finally, COCOMO models for software maintenance will be introduced and validated using industry data sets.

Hypothesis 1*: The measure of the SLOC deleted from the modified modules is not a significant size metric for estimating the effort of software maintenance projects.*

Hypothesis 2: *The productivity ranges of the cost drivers in the COCOMO II model for maintenance are different from those of the cost drivers in the COCOMO II.2000 model.*

Hypothesis 3*: The COCOMO II model for maintenance outperforms the COCOMO II.2000 model when estimating the effort of software maintenance projects.*

Hypothesis 4*: The COCOMO II model for maintenance outperforms the simple linear regression model and the productivity index estimation method.*

2 Related Works

Software cost estimation has attracted tremendous attention from the software engineering research community. A number of studies have been published to address cost estimation related problems, such as software sizing, software productivity factors, cost estimation models for software development and maintenance.

2.1 Software Sizing

Size is one of the most important attributes of a software product. It is a key indicator of software cost and time; it is also a base unit to derive other metrics for software project measurements, such as productivity and defect density. This section describes the most popular sizing metrics and techniques that have been proposed and applied in practice. These techniques can be categorized into code-based sizing metrics and functional size measurements.

2.2 Major Cost Estimation Models

Many estimation models have been proposed and applied over the years. Instead of describing them all, this section provides a brief review of major estimation models that have been developed, continued to be applied, and marketed by respective developers. These models include SLIM, SEER-SEM, PRICE-S, Knowledge Plan, and COCOMO. There are several reasons for this selection. First, they represent the core set of models that was developed in the early 1980's and 1990's. Second, they are still being investigated and used widely in practice and literature. Their long history of extensions and adoptions is proof of their robustness and usefulness. Third, these models perform estimation for a broad range of software development and maintenance activities, covering a number of phases of software lifecycles such as requirements, architecture, implementation, testing, and maintenance.

2.3 Maintenance Cost Estimation Models

Although the area of software maintenance estimation has received less attention as compared to that of new development, given the importance of software maintenance, a number of models have been

introduced and applied to estimating the maintenance costs. These models address diverse sets of software maintenance work, covering, for instance, error corrections, functional enhancements, technical renovations, and reengineering. They can be roughly classified into three types based on the granularity level of the estimation focus: phase-, release-, and task-level maintenance estimation models.

3.The Research Approach

This part presents our research approach and framework to derive the extended COCOMO II size and effort models for software maintenance. Section 3.1 details the 8- step modeling process that is based on the generic COCOMO II modeling methodology discussed in [Boehm 2000b]. The techniques used to calibrate the proposed COCOMO II effort model for software maintenance are discussed in Section 3.2. And Section 3.3 presents the strategies used to validate and compare the performance of estimation models.

3.1 The Modeling Methodology

Figure 3-1 represents the process to be followed to derive the extended COCOMO model for software maintenance.

This process is based on the modeling methodology proposed and applied to building several models in the COCOMO model suite, such as COCOMO II, COQUALMO, CORADMO, and COCOTS [Boehm 2000b].

As the COCOMO model for software maintenance addressed in this study is an extension of the COCOMO II model, Steps 1, 3A, and 4 are performed with a consideration that the model would share most, if not all, of the cost drivers used in the COCOMO II model. Additionally, any step can be revisited if updated information or data would change the final result.

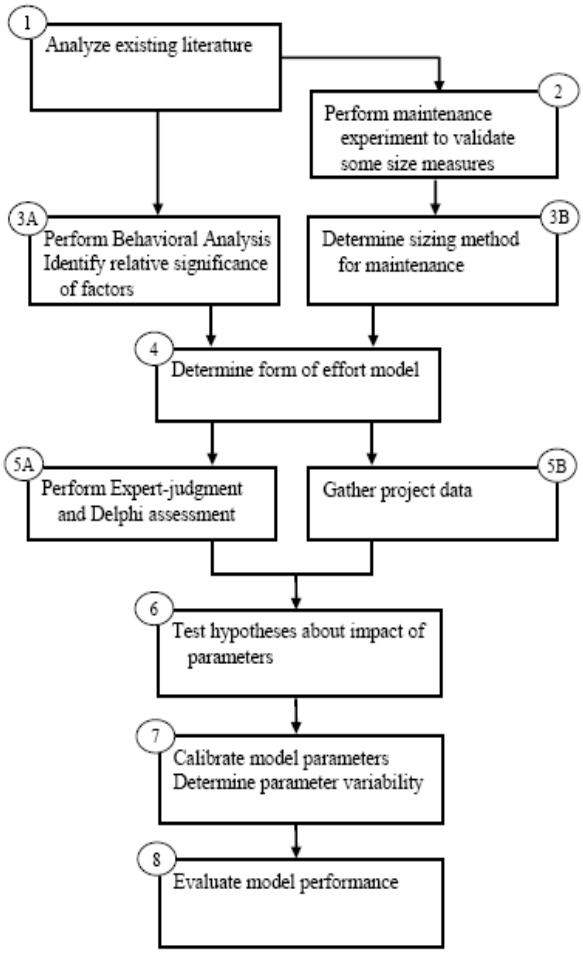

Figure 3-1. The Modeling Process

3.2 The Calibration Techniques

This section describes three calibration techniques, namely ordinary least squares regression, Bayesian analysis, and constrained regression technique, which are applied to calibrating the cost drivers of the model.

3.2.1OrdinaryLeast Squares Regression

Ordinary least squares (OLS) regression is the most popular technique used to build software cost estimation models. In COCOMO, the OLS is used for many purposes, such as analyzing the correlation

between cost drivers and the effort and generating coefficients and their variances during the Bayesian analysis. The ordinary least squares (OLS) estimates for the regression coefficients are obtained by minimizing the sum of square errors. Thus, the response estimated from the regression line minimizes the sum of squared distances between the regression line and the observed response.

3.2.2 The Bayesian Analysis

The Bayesian approach relies on Bayes' theorem to combine the *a priori* knowledge and the sample information in order to produce an *a posteriori* model. In the COCOMO context, the *a priori* knowledge is the expert-judgment estimates and variances of parameter values; the sample information is the data collected from completed projects. To compute the posterior mean and variance of the coefficients, we need to determine the mean and variance of the expert-judgment estimates and the sampling information. Steps 5A and 5B in the modeling process Figure 3-1 are followed to obtain these data.

4.The COCOMO II® Model For Real Time System Software Maintenance

4.1 Software Maintenance Sizing Methods

Size is the most significant and essential factor for the cost estimation model, inclusion of software maintenance cost (see Section 2.3). Thus, we must provide methods to measure the size of work effectively.

Unlike the new development, where the code for core capabilities does not exist, the software maintenance relies on the preexisting code. As a result, the maintenance sizing method must take into account different types of code and other factors, such as the effects of understanding the preexisting code and the quality and quantity of the preexisting code.

Figure 4-1 shows preexisting code as source inputs and delivered code as outputs of the development and maintenance activities.

4.2 COCOMO II® Effort Model For Real Time System Software Maintenance

We first assume that the cost of software maintenance follows the same form of the COCOMO II model. In other words, the model is nonlinear and consists of additive, multiplicative, and exponential components [Boehm and Valerdi 2008].

Furthermore, the cost drivers' definitions and rating levels remain the same except that the Developed for Reusability (RUSE) and Required Development Schedule (SCED) cost drivers were eliminated, and rating levels for the Required Software Reliability

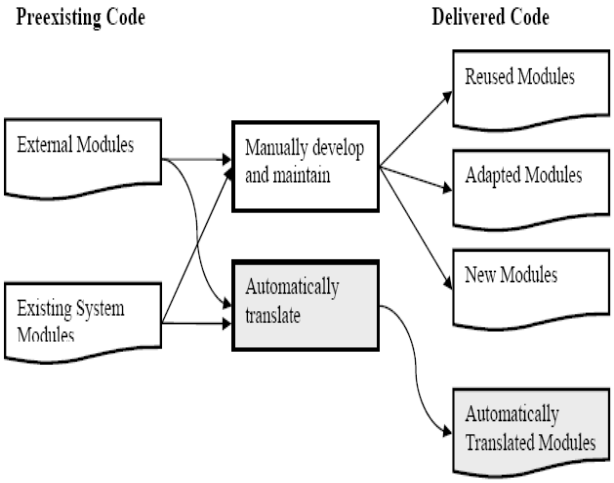

Figure 4.1 Types of Code

(RELY), Applications Experience (APEX), Platform Experience (PLEX), and Language and Tool Experience (LTEX) were adjusted. Details of the changes and rationale for the changes are given as follows.

Elimination of SCED and RUSE

As compared with the COCOMO II model, the *Developed for Reusability* (RUSE) and *Required Development Schedule* (SCED) cost drivers were excluded from the effort model for software maintenance, and the initial rating scales of the *Required Software Reliability* (RELY) cost driver were adjusted to reflect the characteristics of software

maintenance projects. As defined in COCOMO II, the RUSE cost driver "accounts for additional effort needed to develop components intended for reuse on current or future projects." This additional effort is spent on requirements, design, documentation, and testing activities to ensure the software components are reusable. In software maintenance, the maintain team usually adapts, reuses, or modifies the existing reusable components, and thus, this additional effort is less relevant as it is in development. Additionally, the sizing method already accounts for additional effort needed for integrating and testing the reused components through the *IM* parameter.

5. Research Results

5.1The Controlled Experiment Results

Hypothesis 1 states that the SLOC deleted from the modified modules is not a significant size metric for estimating the maintenance effort. One approach to testing this hypothesis is to validate and compare the estimation accuracies of the model using the deleted SLOC and those of the model not using the deleted SLOC. Unfortunately, due to the effects of other factors on the software maintenance effort, this approach is impractical. Thus, the controlled experiment method was used as an approach to testing this hypothesis. In a controlled experiment, various effects can be isolated.

We performed a controlled experiment of student programmers performing maintenance tasks on a small C++ program [Nguyen 2009, Nguyen 2010]. The purpose of the study was to assess size and effort implications and labor distributions of three different maintenance types and to describe estimation models to predict the programmer's effort on maintenance tasks.

5.1.1 Experiment Results

Maintenance time was calculated as the duration between finish and start time excluding the interruption time if any. The resulting timesheet had a total of 490 records totaling 4,621 minutes. On average, each participant recorded 19.6 activities with a total of 192.5 minutes or 3.2 hours. We did not include the acceptance test effort because it was done independently after the participants completed and submitted their work. Indeed, in the real-world situation the acceptance test is usually performed by customers or independent teams, and their effort is often not recorded as the effort spent by the maintenance team. The sizes of changes were collected in terms of the number of SLOC added, modified, and deleted by comparing the original with the modified version. These SLOC values were then adjusted using the proposed sizing method to obtain equivalent SLOC. We measured the SLOC of task-relevant code fragments (*TRCF*) by summing the size of all affected methods. As a SLOC is corresponding to one logical source statement, one SLOC modified can easily be distinguished from a combination of one added and one deleted.

The first three charts in Figure 5-1 show the distribution of effort of four different activities by participants in each group. The forth chart shows the overall distribution of effort by combining all three groups. Participants spent the largest proportion of time on coding, and they spent much more time on the isolation activity than the testing activity. By comparing the distribution of effort among the groups, we can see that proportions of effort spent on the maintenance activities vary vastly among three groups.

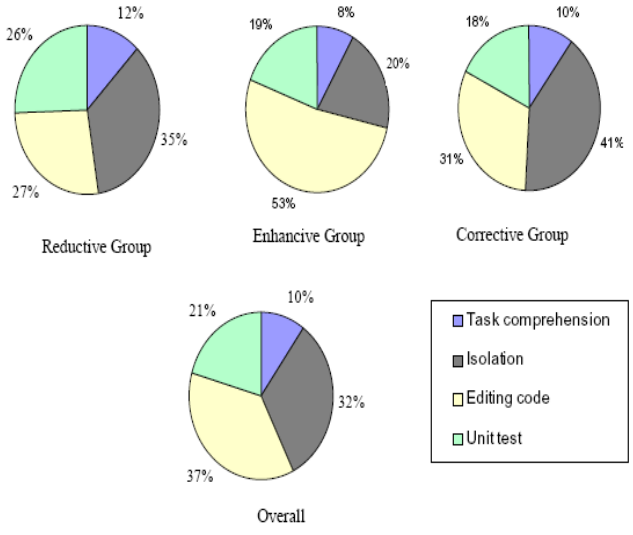

Figure 5-1. Effort Distribution

6 Future Work

There are number of directions for future work that are worth exploring. These directions involve further calibrating the model, extending it to other types of maintenance, to iterative and agile software development methods, and improving the constrained regression approaches. Software estimation modeling is a continual process. As advancements in technologies, software processes, languages, and supporting tools have accelerated rapidly, the productivity of software projects increases over the years [Nguyen 2010b]. The software estimation model has to be recalibrated and extended to more reflect more closely the software development and maintenance practice. Moreover, three organizations from which the data set was collected are definitely not representative of the software industry. Thus, the model calibrated using this data set may not reflect effectively the *true* practice of software maintenance. However, considering this work as a step in making COCOMO a better model for estimating the cost of software maintenance, the maintenance model should be calibrated with data points from a variety of sources. Understandably, collecting software cost related metrics from industry is difficult. Due to the sensitivity of these metrics, organizations are reluctant to share. It is even more challenging to collect maintenance data since it requires more detailed sizing metrics and organizations often do not have a software maintenance process as rigorous as the development counterpart to mandate the data collection and analysis practice.

References

[1] Abrahamsson P., Moser R., Pedrycz W., Sillitti A., Succi G. (2007), "Effort Prediction in Iterative Software Development Processes -- Incremental Versus Global Prediction Models", Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM)

[2] Abran A. and Maya M. (1995), "A Sizing Measure for Adaptive Maintenance Work Products," Proceedings of the 11th International Conference on Software Maintenance (ICSM), pp.286-294

[3]Abran A., St-Pierre D., Maya M., Desharnais J.M. (1998), "Full function points for embedded and real-time software", Proceedings of the UKSMA Fall Conference, London, UK, 14.

[4] Abran A., Silva I., Primera L. (2002), "Field studies using functional size measurement in building estimation models for software maintenance", Journal of Software Maintenance and Evolution, Vol 14, part 1, pp. 31-64

[5] Albrecht A.J. (1979), "Measuring Application Development Productivity," Proc. IBM Applications Development Symp., SHARE-Guide, pp. 83-92.

[6] Albrecht A.J. and Gaffney J. E. (1983) "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November

[7] Banker R., Kauffman R., and Kumar R. (1994), "An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment," Journal of Management Information System.

[8] Basili V.R., (1990) "Viewing Maintenance as Reuse-Oriented Software Development," IEEE Software, vol. 7, no. 1, pp. 19-25, Jan.

[9] Basili V.R., Briand L., Condon S., Kim Y.M., Melo W.L., Valett J.D. (1996), "Understanding and predicting the process of software maintenance releases," Proceedings of International Conference on Software Engineering, Berlin, Germany, pp. 464–474.

[10] Basili V.R., Condon S.E., Emam K.E., Hendrick R.B., Melo W. (1997) "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components". Proceedings of the 19th International Conference on Software Engineering, pp.282-291

First Author Sugandha chakraverti is active member of IEEE and IEEE computer society she is doing her P.hd. in Computer

Science and Engineering at MBU Solan HP India. She has published many national and International paper and her area of interests are Software Engineering, Real Time System, Distributed system and Data Mining.

Second Author Prof. Sheo Kumar has 15 years of Industry and Teaching experience. He has published many no of research article, papers in national and international journal. He is working as Associate Professor at IEC-CET Greater Noida UP India and is a Research Scholar of MBU Solan HP India. His area of interests are Distributed system, Artificial intelligence and Neural Networks. He is also an active member of IEEE.

Third Author Dr S.C.Agarwal is Ph.d. from IIT-R india in Mathematics for Computation and working as a Director at KCCEC Greater Noida UP India. He has around 30 years of Experience as Academician and Researcher in leading organization of india.

Fourth Author Ashish Kumar Chakraverti has 7 years of Teaching experience. He did his B.Tech. and M.Tech in CSE. He

has published many national and international research paper. He is working as Assistant professor at BBDIT Ghaziabad UP

India and his area of interest is computer networks, Mathematical computer science and Compiler design.