ISSN:2320-0790

An International Journal of Advanced Computer Technology

Evaluation of Calibration Techniques to Build Software Cost Estimation Models

Safia Yasmeen[1], Prof.Dr.G.Manoj Someswar[2]

1. Research Scholar, Mahatma Gandhi Kashi Vidyapith, Varnasi, U.P., India
2. Research Supervisor, Mahatma Gandhi Kashi Vidyapith, Varnasi, U.P., India

Abstract: This research paper 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.

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.

Keywords: Ordinary Least Squares Regression, Bayesian Analysis, Multiple Regression Technique, Develop for Reusability(RUSE), Constrained Multiple Regression Technique, Constrained Minimum Sum of Square Errors(CMSE), Constrained Minimum Sum of Absolute Errors, Required Development Schedule(SCED)

INTRODUCTION

Suppose that a [illegible] sponse is effort and p predictors are, for example, size and cost drivers. Let $(x_{i1}, x_{i2}, ..., x_{ip}), i = 1, 2, ..., N,$ be the vector of p predictors, and *yi* be the response for the *i*th observation. The model for multiple linear regression can be expressed as

(Eq. 3-1)

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \varepsilon_i$$

Where $\beta_0, \beta_1, ..., \beta_p$ are the regression coefficients, and e, is the error term for the *i*th observation. The corresponding prediction equation of (Eq. 3-1) is

(Eq. 3-2)

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$$

where $\beta_0, \beta_1, ..., \beta_p$ are the estimates of coefficients, and y is the estimate of response for the *i*th observation.

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.[1]

Although regression is a standard method for estimating software cost models, it faces some major challenges. The model may be over- fitted. This occurs when unnecessary predictors remain in the model. With software cost data, some of the predictors are highly correlated. [2] Such co-linearity may cause high variances and co-variances in coefficients and result in poor predictive performance when one encounters new data. We can sometimes ameliorate these problems by reducing the number of

predictor variables. By retaining only the most important variables, we increase the interpretability of the model and reduce the cost of the data collection process. As empirical evidence of this effectiveness, Chen et al. report that the reduced-parameter COCOMO models can yield lower prediction errors and lower variance.

THE BAYESIAN ANALYSIS

The Bayesian approach to calibrating the COCOMO II model was introduced for the purpose

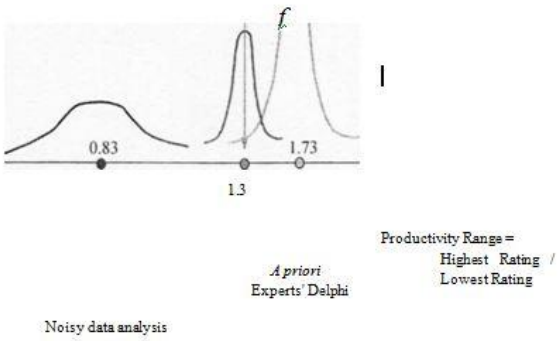


Figure 1: A Posteriori Bayesian Update in the Presence of Noisy Data RUSE

The Bayesian approach calculates the posterior mean b and variance Var(b) of the coefficients as:

(Eq. 3-3)

$$b^{**} = \left(\frac{1}{s^2}X'X + H^*\right)^{-1} \times \left(\frac{1}{s^2}X'Xb + H^*b^*\right)$$

of this study. This study produced a set of constants and cost drivers officially published in the COCOMO II book [Boehm 2000b]. Since then, the Bayesian approach has been used in a number of calibrations of the model using multiple COCOMO data sets.[3]

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. Figure 1 shows the productivity range (the ratio between the highest and lowest rating values) of the RUSE (Develop for Reusability) cost driver obtained by combining a priori expert-judgment estimate and data-determined value.

Where,

- X and S2 are the matrix of parameters and the variance of the residual for the sample data, respectively.
-

$$Var(b^{**}) = \left(\frac{1}{s^2} X^{'} X + H^{*} \right)^{-1} \quad \text{(Eq. 3-4)}$$

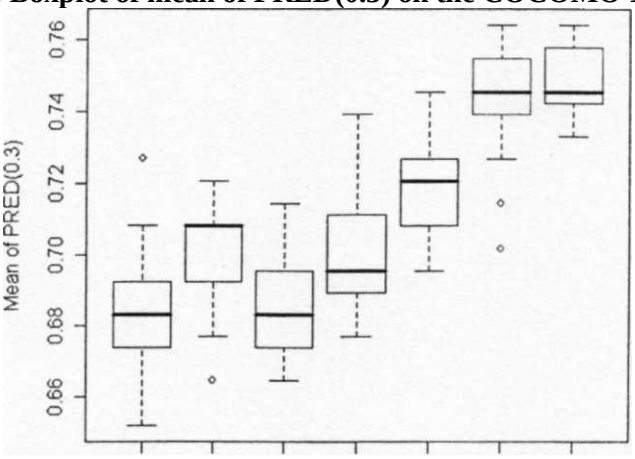
H and b are the inverse of variance and the mean of the prior information (expert-judgment estimates), respectively.

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 1 are followed to obtain these data.

A CONSTRAINED MULTIPLE REGRESSION TECHNIQUE

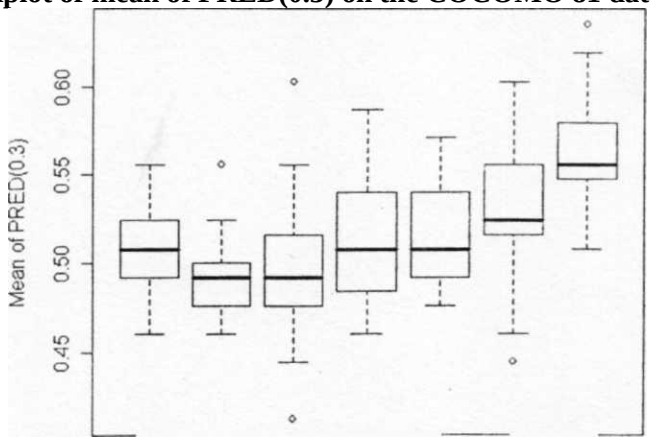
In this research work, we proposed the regression techniques to calibrate the COCOMO model coefficients. The technique estimates the model coefficients by minimizing objective functions while imposing model constraints. The objective functions represent the overall goal of the model, that is, to achieve high estimation accuracies.[4] The constraints can be considered subordinate goals, or the priori knowledge, about the model. We validated the technique on two data sets used to construct COCOMO 81 and COCOMO 11.2000. The results indicate that the technique can improve the performance of the COCOMO II model (see Figure 2 and Figure 3). On both COCOMO 11.2000 and COCOMO 81 data sets, the constrained techniques CMAE and CMRE were found to outperform the other techniques compared. With this finding, we will apply this technique to calibrate the model and compare the calibration results obtained by this technique with those of the Bayesian analysis.

Figure 2: Boxplot of mean of PRED(0.3) on the COCOMO 11.2000 data set



Lasso Ridge Stepwise OLS CMSE CMAE CMRE

Figure 3: Boxplot of mean of PRED(0.3) on the COCOMO 81 data set



H --- 1---- r ----1 --- 1 i--- r

Lasso Ridge Stepwise OLS CMSE *CMAE CMPE*

The technique consists of building three models for calibrating the coefficients in Equation (Eq. 4-12). These models are based on three objective functions MSE, MAE, and MRE that have been well investigated and applied to building or evaluating cost estimation models. MSE is a technique minimizing the sum of square errors, MAE minimizing the sum of absolute errors, and MRE minimizing the sum of relative errors. The models examined include:

(1) Constrained Minimum Sum of Square Errors (CMSE)

(Eq. 3-5)

$$\text{Minimize} \quad \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\text{subject to } MRE_i \le c \text{ and } \hat{\beta}_j \ge 0,\ i = 1$$

(2) Constrained Minimum Sum of Absolute Errors (CMAE):

(Eq. 3-6)

$$\text{Minimize} \quad \sum_{i=1}^{N}|y_i - \hat{y}_i|$$

$$\text{subject to } MRE_i \le c \text{ and } \hat{\beta}_j \ge 0,\ i = 1,\ldots, N, j = 0,\ldots, p$$

(Eq. 3-7)

$$\text{Minimize} \quad \sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

$$\text{subject to } MRE_i \le c \text{ and } \hat{\beta}_j \ge 0,\ i = 1,\ldots, N, j = 0,\ldots, p$$

Where, c > 0 is the turning parameter controlling the upper bound of MRE for each estimate, and MRE, is the magnitude of relative error of the estimate z'th.

Estimating fJ0,J3x,.,.,J3p in Equations (Eq. 3-5), (Eq. 3-6), and (Eq. 3-7) is an optimization problem. Equation (Eq. 3-6) is a quadratic programming problem.

Also, Equations (Eq. 3-5) and (Eq. 3-7) can be transformed to a form of the linear programming. A procedure for this transformation is done as part of our research work. In this study, we use quadratic and linear programming solvers (quadprog7 and 8* IpSolve) provided in the R statistical packages to estimate the coefficients.

One of the advantages of this technique is that the priori knowledge can be included in the regression models in the form of constraints to adjust the estimates of coefficients. The constraints can be any functions of the model parameters that are known prior to building the model. For example, in COCOMO the estimates of coefficients should be non-negative (e.g., an increase in the parameter value will result in an increase in effort). As the constraints are applied, the technique can effectively prune parameters that are negative while adjusting other parameters to minimize the objective function.

EVALUATION STRATEGIES

MODEL ACCURACY MEASURES

MMRE and PRED are the most widely used metrics for evaluating the accuracy of cost estimation models. These metrics are calculated based on a number of actuals observed and estimates generated by the model. They are derived from the basic magnitude of the relative error MRE, which is defined as (Eq. 3-8) where y, and j>. are the actual and the estimate of the ith observation, respectively. Because yt is log-transformed, we calculate the MRE, using (Eq. 3-9).

$$MRE_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

(Eq. 3-8)

$$MRE_i = \left| 1 - e^{\hat{y}_i - y_i} \right|$$

(Eq. 3-9)

The mean of MRE of N estimates is defined as (Eq. 3-10). As every estimate is included in calculating MMRE, extreme values of MRE can significantly affect MMRE. To handle this problem, another important criterion used for model evaluation is PRED. PRED(/) is defined as the percentage of estimates, where MRE is not greater than /, that is PRED(/) = k/n, where k is the number of estimates with MRE falling within /, and n is the total number of estimates.[5] We can see that unlike MMRE, PRED(/) is insensitive to errors greater than /. Another accuracy measure that has been often reported in the software estimation research is the median of the magnitude of relative errors (MdMRE). Unlike MMRE, the MdMRE measure provides information about the concentration of errors and is not affected by extreme errors. Using these measures as model comparison criteria, one model is said to outperform another if it has lower MMRE. MdMRE, and higher PRED(/). In this research, the results are reported and compared mainly using PRED(0.3) and MMRE. This measure is considered a standard in reporting COCOMO calibration and model improvement in the previous studies. In addition, to allow comparisons between the models investigated in this study with others, PRED(0.25), PRED(0.50), and MdMRE measures are also reported.

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i$$

(Eq. 3-10)

CROSS-VALIDATION

The most important criterion for rejection or acceptance of a cost estimation model is its ability to predict using new data. Ideally the prediction error of a new cost model is calculated using data from future projects. This approach, however, is usually impossible in practice because new data is not always available at the time the model is developed. Instead, model developers have to use the data that is available to them for both constructing and validating the model. This strategy is usually referred to as cross-validation.

While many cross-validation approaches have been proposed, the most common are a simple holdout strategy and a computer-intensive method called K-fold cross validation. The holdout approach

splits the dataset into to two distinctive subsets: training and test sets. The training set is used to fit the model and the test set provides estimates of the prediction errors. K-fold cross validation divides the data into K subsets. Each time, one of the K subsets is used as the test set and the other K-l subsets form a training set. [6] Then, the average error across all K trials is computed. K-fold cross validation avoids the issue of overly-optimistic results for prediction accuracy. This technique enables the user to independently choose the size of each test set and the number of trials to use for averaging the results. The

Step 1. Randomly split the dataset into K subsets

Step 2. For each i= 1,2,K:

$$MMRE_i = \frac{1}{P}\sum_{j=1}^{P}\left|\frac{y_j - \hat{y}_{\bullet j}}{y_j}\right|$$

(Eq. 3-11)

MMREt is calculated as (Eq. 3-11) Where, P is the number of observations in the /'th subset, y,, is the estimate of they'th observation in the /th subset, and / = 0.3, 0.25, 0.2, and 0.1.

The special case where K = N is often called as leave-one-out cross-validation (LOOC). In this method, the training set that consists of N - 1 observations is used to build the model to test the remaining observation. LOOC appears to be a preferred cross-validation method used for validating the performance of software estimation models. One possible reason is that software estimation data is scarce, and thus, models cannot afford to leave more data points out of the training set. Another reason is that the approach reflects the reality in which all available data of an organization is used to calibrate the model for future projects. Therefore, LOOC is used in this study.

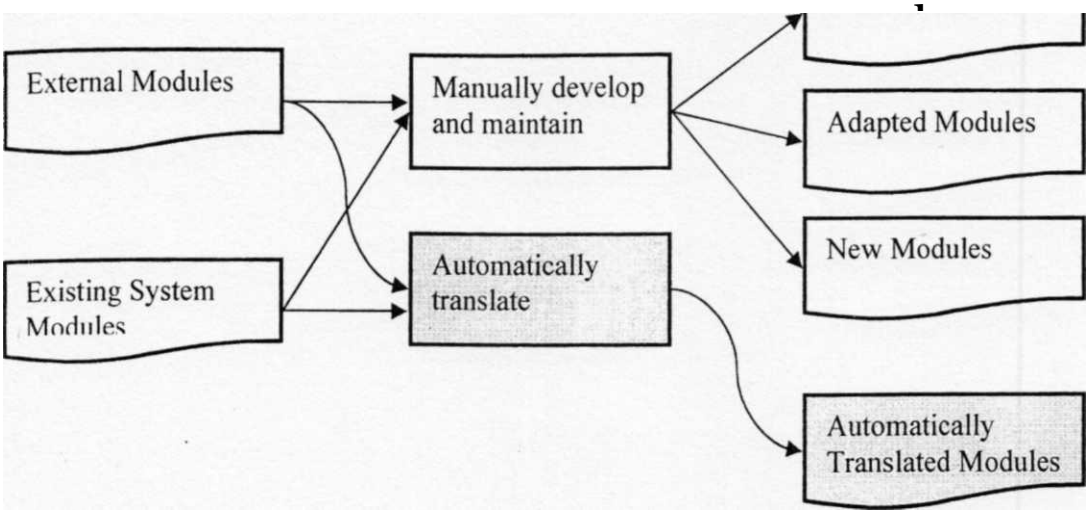
variance of the resulting estimate is reduced as K is increased. The disadvantage of this method is that the training algorithm has to be rerun K times, resulting in computational effort. The K-fold cross-validation procedure can be described in the following three steps:

*Build the model with the /th subset of the data removed.

*Predict effort for observations in the /th subset.

*Calculate MMREt and PRED(l)i for the fth subset.

Figure 4: Types of Code



■ External modules: the code taken from a source other than the system to be maintained. They can be proprietary or open-source code.

■ Preexisting system modules: the code of the system to be upgraded or maintained.

■ Reused modules: the preexisting code that is used as a black-box without modifications.

■ Adapted modules: the code that is changed from using the preexisting code. The preexisting code is used as a white-box in which source lines of code are added, deleted, modified, and unmodified.

■ New modules: the modules newly added to the updated system.

■ Automatically translated modules: the code obtained from using code translation tools to translate the preexisting code for use in the updated system. In COCOMO, the automatically translated code is not included in the size of the maintenance and reuse work. Instead, the effort associated with the automatically translated code is estimated in a separate model different from the main COCOMO effort model. In the COCOMO estimation model for software maintenance, we also exclude the

automatically translated code from the sizing method and effort estimation.

THE COCOMO II REUSE AND MAINTENANCE MODELS

COCOMO II provides two separate models for sizing software reuse and software maintenance. The reuse model is used to compute the equivalent size of the code that is reused and adapted from other sources. The reuse model can also be used for sizing major software enhancements.[7] On the other hand, the maintenance model is designed to measure the size of minor enhancements and fault corrections.

The COCOMO II reuse sizing model was derived on the basis of experience and findings drawn from previous empirical studies on software reuse costs. We performed an analysis of reuse costs of reused modules in the NASA Software Engineering Laboratory, indicating nonlinear effects of the reuse cost function (Figure 5). This describes a formula to represent the number of interface checks required in terms of the number of modules modified and the total number of software modules, showing that the relationship between the number of interface checks required and the number of modules modified is nonlinear. The cost of understanding and testing the existing code could, in part, cause the nonlinear effects. We found that the effort required to understand the software be modified takes 47 percent of the total maintenance effort.

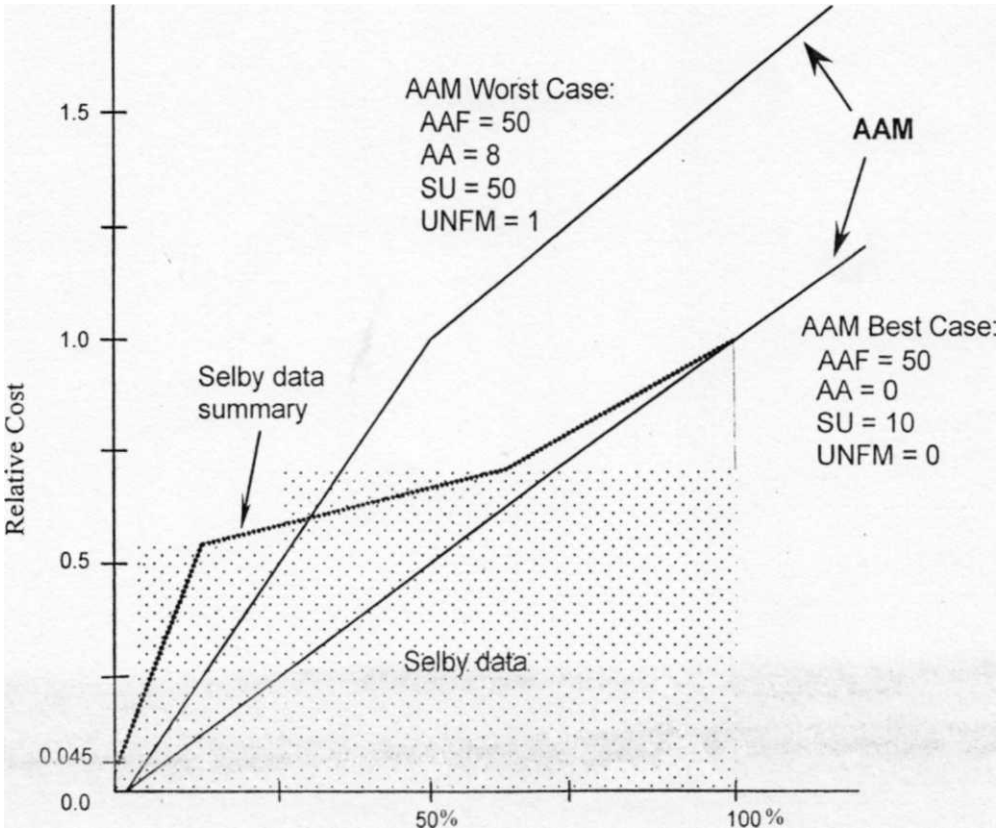


Figure 5: Nonlinear Reuse Effects

The COCOMO II reuse model computes the equivalent SLOC for enhancing, adapting, and reusing the pre-existing software. This model takes into account the amount of software to be adapted, percentage of design modified (DM), the percentage of code modified (CM), the percentage of integration and testing required (IM), the level of software understanding (SU), and the programmer's relative unfamiliarity with the software (UNFM). The model is expressed as:

Equivalent KSLOC = Adapted KLOC $*(1-\frac{AT}{100})*AAM$ **(Eq. 4-1)**

Where,

$$AAF = 0.4*DM + 0.3*CM + 0.3*IM \quad \textbf{(Eq. 4-2)}$$

$$AAM = \begin{cases} \frac{AA + AAF(1+0.02*SU*UNFM)}{100}, & for\ AAF \le 50 \\ \frac{AA + AAF + SU*UNFM}{100}, & for\ AAF > 50 \end{cases} \quad \textbf{(Eq. 4-3)}$$

AT is the total Automatic Translation code; AA is the degree of Assessment and Assimilation; AAF is the Adaptation Adjustment Factor representing the amount of modification; and MM stands for Adaptation Adjustment Multiplier. The factors SU and UNFM in the model are used to adjust for software comprehension effects on the adaptation and reuse effort, reflecting the cost of understanding the software to be modified [Parikh and Zvegintzov 1983, Nguyen 2009, Nguyen 2010]. Figure 5 shows the region of possible AAM values specified by the parameters AAF, AA, SU, and UFNM.

Software Understanding (SU) measures the degree of understandability of the existing software (how easy it is to understand the existing code). The rating scale ranges from Very Low (very difficult to understand) to Very High (very easy to understand). SU specifies how much increment to Equivalent SLOC (ESLOC) is needed if the programmer is new to the existing code. We prepared a table that best describes the numeric SU rating scale for each rating level for the purpose of our research work. At the

rating of Very Low, the developers would spend 50% of their effort for understanding the software for an equivalent amount of code.

Programmer Unfamiliarity (UNFM) measures the degree of unfamiliarity of the programmer with the existing software. This factor is applied multiplicatively to the software understanding effort increment.[8]

Assessment and Assimilation (AA) is the degree of assessment and assimilation needed to determine whether a reused software module is appropriate to the system, and to integrate its description into the overall product description. AA is measured as the percentage of effort required to assess and assimilate the existing code as compared to the total effort for software of comparable size.

Equivalent SLOC is equivalent to SLOC of all new code that would be produced by the same amount of effort. Thus, Equivalent SLOC would be equal to new SLOC if the project is developed from scratch with all new code.

The COCOMO II Maintenance Sizing Model

Depending on the availability of the data, several means can be used to calculate the size of maintenance. One way is to determine the maintenance size based on the size of the base code (BCS), the percentage of change to the base code named Maintenance Change Factor (MCF), and an adjustment factor called Maintenance Adjustment Factor MAF).

Size = BCS x MCF xMAF **(Eq. 4-4)**

Alternatively, COCOMO can measure the size of maintenance based on the size of added and modified code, and adjusts it with the MAF factor. MAF is adjusted with the SU and UNFM factors from the Reuse model. That is,

MAF = 1 + (SUx UNFM/100) **(Eq. 4-5)**

Thus, Size = (Added + Modified) x [1 + SUxUNFM/100] **(Eq. 4-6)**

The maintenance size measure is then used as an input to the COCOMO II models to generate the effort and schedule estimates. The COCOMO II model assumes that the software maintenance cost is influenced by the same set of cost drivers and their ratings as is the development cost, with some exceptions noted above.

A UNIFIED REUSE AND MAINTENANCE MODEL

This research work presents our proposed sizing model for software maintenance that unites and improves the existing COCOMO II reuse and maintenance models. This model is proposed to address the following limitations of the existing models.

■ The reuse model would underestimate code expansion. The software system grows over time at a significant rate as the result of continuing functional attributes.

■ DM is the percentage of the design modification made to the analysis and design artifacts of the preexisting software affected by the changes for the new release or product. DM does not include the design related to the code expansion (e.g., new classes and methods) because the code expansion is taken into account by CM. The DM value ranges from 0 to 100%.[9]

■ CM is the percentage of code added, modified, and deleted relative to the size of the preexisting modules affected by the changes for the new release or product. In other words, CM is equal to the sum of SLOC added, modified, and deleted divided by the total SLOC of the preexisting code. It includes code expansions, which may go beyond the size of the preexisting code, and thus CM can exceed 100%.

■ IM is the percentage of integration and test needed for the preexisting modules to be adapted into the new release or product, relative to the normal amount of integration and test for the preexisting modules affected by the changes. IM may exceed 100% if the integration and test is required for other parts of the system that are related to the changes or some special integration and test is required to validate and verify the whole system. Like DM, IM does not include the integration and test for the code expansion as CM accounts for this effect.

Using the parameters DM, CM, IM, SU, and UNFM,.the formulas used to compute AAF and AAM are presented as:

AAF = 0A* DM + CM + 03* IM

(Eq. 4-7)

AA + AAF +

1- 1-

AAF

*SU*UNFM

AAM - \

J

100

100

AA + AAF + SU * UNFM

100

if AAF < 100

if AAF > 100

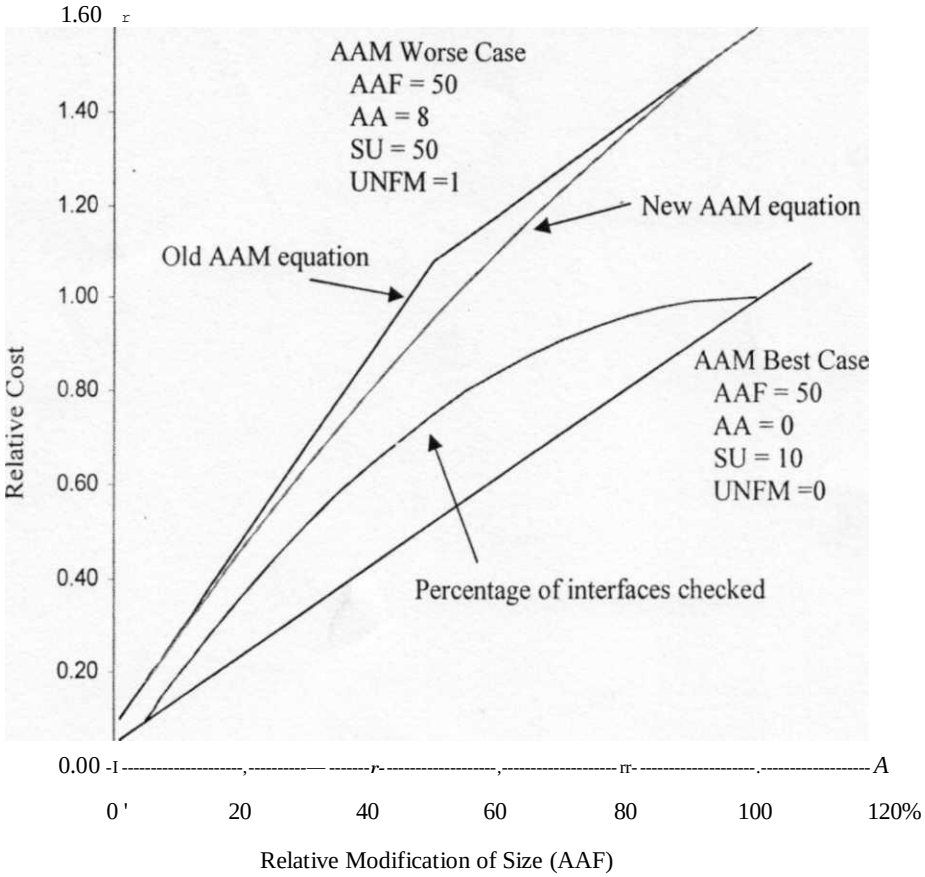
(Eq. 4-8)

Although Equations (Eq. 4-7) and (Eq. 4-8) are based on those of the COCOMO reuse model, they are different in several ways. In the AAF formula, the CM coefficient is 1 instead of 0.3 while the coefficients for DM and IM are the same as those in Equation (Eq. 4-2). This change reflects the new definition of CM, which accounts for code expansion and considers that a SLOC modified or deleted is same as a SLOC added. AAF represents the equivalent relative size of the changes for the new release or product, and its value is greater than 0% and may exceed 100%. For AAM, Equation (Eq. 4-3) presents the AAM curve, which consists of two straight lines joining at AAF = 50%, which is less intuitive because the breakage at AAF = 50% is not demonstrated empirically or theoretically. The new AAF Equation (Eq. 4-7) smoothes the curve when AAF < 100%. The difference between the old and new AAF curves is shown in Figure 4-3. This difference is most significant when AAF is close to 50%: the old AAF overestimates AAM as compared to the new AAF. The difference decreases in the direction moving from the AAM worse case to the AAM best case. [10] As a result, the two equations produce the same AAM valuesfor the AAM best case.

Now, let's show that the smoothed curve of new AAM Equation (Eq. 4-8) can be derived while maintaining the nonlinear effects discussed above. First, it is important to

note that $AAMis$ proposed as a model representing the nonlinear effects involved in the module interface checking and testing interfaces directly affected or related to the changes. It, therefore, takes into account not only the size of the changes but also the size of the modules to be changed. Second, we assume that there is one interface between two modules.

Figure 4-3. AAM Curves Reflecting Nonlinear Effects



Let n be the number of modules of the system and x be the percentage of modification. The number of interfaces among n modules is

$$\frac{n^2}{2} - \frac{n}{2} \quad (\text{for } n \gg 0)$$

As the number (percentage) of unmodified modules is $100 - x$, the number of interfaces in the unmodified modules can be approximated as $(100 - x)'$ The number of interfaces remained to be checked is ——^———. Therefore, the percentage of interfaces to be check is $1 - \left(\frac{x}{100}\right)^2$. Here, x is the percentage of modification, which represents AAF, or we get $1 - \frac{AAF}{100}$ as the percentage of code that requires checking. The quantity $1 - 1 - \frac{AAF}{100} : SU * UNFM$ in Equation (Eq. 4-8) accounts for the effects of understanding of the interfaces to be checked.

Although different in the form, the percentage of code that requires checking $1 - \frac{AAF}{100}$ is close to Gerlich and Denskat [1994], which demonstrates that the number of interface checks requires, N, is

$$N = k * (m - k) + k * \qquad \text{(Eq. 4-9)}$$

where k and m are the number of modified modules and total modules in the software, respectively.

The unified model classified the delivered code into three different module types, eused, adapted, and new as described above. Considering the differences among these types, we use different parameters and formulas to measure the size of each type separately. The size of deleted modules is not included in the model.

New Modules

The module is added to the system; thus, its size is simply the added KSLOC count (KSLOC added) without considering the effects of module checking and understanding.

Adapted Modules

The equivalent size (EKSLOC adapted) is measured using the size of the preexisting modules to be adapted (AKSLOQ and the AAM factor described in Equation (Eq. 4-8).

EKSLOC adapted = AKLOC * AAM Reused Modules

As these modules are not modified, the DM, CM, SU, and UNFM are all zero. Thus, the equivalent KSLOC (EKSLOC) of the reused modules is computed as

EKSLOCreused = RKSLOC * AAM,

where RKSLOC is the KSLOC of the reused modules, and AAM is computed as

AAM = (AAreused + 0.3 * IMreUsed) / 100 **(Eq.4-9)**

AA reused is the degree of assessment and assimilation needed to determine the modules relevant for reuse in the maintained system. It is measured as the percentage of effort spent to assess and assimilate the existing code versus the total effort needed to write the reused modules from scratch.[11]

Finally, the equivalent SLOC is computed by the formula:

EKSLOC = KSLOC adM+EKSLOC adapled+EKSLOC m,sed **(Eq.4-10)**

COCOMOII Effort Model for 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** (RELY), **Applications-Experience** (APEX), **Platform Experience** (PLEX), and **Language and Tool Experience** (LTEX) were adjusted. [12] 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 entire the software components are reusable. In software maintenance, the maintenance team usually adapts, reuses, or modifies the existing reusable components,

and thus their additional effort is less relevant as it is in development.[13]

Additionally, the sizing method already accounts for additional effort needed for integrating and testing the reused components through the IM parameter.

Table 4-1: Maintenance Model's Initial Cost

Drivers Scale Factors

PREC Precedentedness of Application

FLEX Development Flexibility

RESL Risk Resolution

TEAM Team Cohesion

PMAT Equivalent Process Maturity Level

Effort Multipliers

Product Factors

RELY Required Software Reliability

DATA Database Size

CPLX Product Complexity

DOCU Documentation Match to Life-Cycle Needs

Platform Factors

TIME - Execution Time Constraint, STOR - Main Storage Constraint, PVOL - Platform Volatility Personnel Factors, ACAP - Analyst Capability, PCAP - Programmer Capability, PCON - Personnel Continuity, APEX - Applications Experience, LTEX - Language and Tool Experience, PLEX - Platform Experience Project Factors.[14]

TOOL Use of Software Tools

SITE Multisite Development

In COCOMO II, the SCED cost driver "measures the schedule constraint imposed on the project team developing the software."[15] The ratings define the percentage of schedule compress sd or extended from a Nominal rating level. According to COCOMO II, schedule compressions require extra effort while schedule extensions do not, and thus, ratings above than Nominal, which represent schedule extensions, are assigned the same value, 1.0, as Nominal. In software maintenance, the schedule constraint is less relevant since the existing system is operational and the maintenance team can produce quick fixes for urgent requests rather than accelerating the schedule for the planned release.

Adjustments for APEX, PLEX, LTEX and RELY

The personnel experience factors (APEX, PLEX and LTEX) were adjusted by increasing the number of years of experience required for each rating. That is, if the maintenance team has an average of 3 years of experience then the rating is Nominal while in COCOMO II the rating assigned for this experience is High.[16] The ratings of APEX, PLEX, and LTEX are shown in Table 4-2. The reason for this adjustment is that the maintenance team in software maintenance tends to remain longer in the same system than in the development. More often, the team continues to maintain the system after they develop it.

Table 4-2: Ratings of Personnel Experience Factors (APEX, PLEX, LTEX)

	Very Low	Low	Nominal	High	Very High
APEX, PLEX, LTEX	< 6 months	1 year	3 years	6 years	12 years

RELY is "the measure of the extent to which the software must perform its intended function over a period of time". In software maintenance, the LY rating values are not monotonic, i.e., they do not only increase or decrease when e RELY rating increases from Very Low to Extra High (see Table 4-3).

The very Low Multiplier is higher than the Nominal 1.0 multiplier due to the extra effort in extending and debugging sloppy software, the Very High multiplier is higher due to the extra effort in CM, QA, and V&V to keep the product at a Very High RELY level.

Table 4-3. Ratings of RELY

	Very Low	**Low**	**Nominal**	**High**	**Verv High**
RELY	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable	high financial loss	risk to human life
Initial multiplier	1.23	1.10	1.0	0.99	1.07

The following will describe the effort form, parameters, and the general transformation technique to be used for the model calibration. The effort estimation model can be written in the following general nonlinear form

$$PM = A*Size^{E} * \prod EM_i \qquad \text{(Eq.4-11)}$$

Where

PM - effort estimate in person months *A*

= multiplicative constant

Size - estimated size of the software, measured in KSLOC. Increasing size has local additive effects on tht effort. Size is referred to as an additive factor.

EM - effort multipliers. These factors have global impacts on the cost of the overall system.

E = is defined as a function of scale factors, in the form of E = B +]T/?,S7v.

Similar to the effort multipliers, the scale factors have global effects across the system but their effects are associated with the size of projects. They have more impact on the cost of larger-sized projects than smaller-sized projects.[17]

From Equation (Eq. 4-11), it is clear that we need to determine the numerical values of the constants, scale factors, and effort multipliers. Moreover, these constants id parameters have to be tamed into historical data so that the model better reflects the effects of the factors in practice and improves the estimation performance. This process is often referred to as calibration.

As Equation (Eq. 4-11) is nonlinear, we need to linearize it by applying the natural logarithmic transformation:

(Eq. 4-12)

log(PM) = fy + Pi \og{Size) + SF, \og(Size) + ...+

(Eq. 4-13)

p6 SF5 \og(Size) + p, \og(EM,) + ... + fa \og(EMl7)

Equation (Eq. 4-12) is a linear form and its coefficients can be estimated using a typical multiple linear regression approach such as ordinary least squares regression. This is a typical method that was used to calibrate the model coefficients and constants. Applying a calibration technique, we can obtain the estimates of coefficients in Equation (Eq. 4-12). The estimates of coefficients are then used to compute the constants and parameter values in equation (Eq. 4-13).

RESEARCH 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. 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.

REFERENCES

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

2. 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

3. Boehm B.W. (1981), "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ, 1981.

4. Boehm B.W. (1988), "Understanding and Controlling Software Costs", IEEE Transactions on Software Engineering.

5. Boehm B.W., Royce W. (1989), "Ada CCCOMO and Ada Process Model," Proc. Fifth COCOMO User's Group Meeting, Nov.

6. Chulani S. (1999), "Bayesian Analysis of Software Cost and Quality Models", PhD Thesis, The University of Southern California.

7. Clark B., Chulani S., and Boehm B.W., (1998), "Calibrating the COCOMO II Post Architecture Model," International Conference on Software Engineering, April.

8. Fioravanti F., Ne: i P., Stortoni F. (1999), "Metrics for Controlling Effort During Adaptive Maintenance of Object Oriented Systems," IEEE International Conferenc; on Software Maintenance (ICSM'99), 1999

9. IFPUG (2004), "IFPUG Counting Practices Manual - Release. 4.2," International Function Point Users Group, Princeton Junction, NJ.

10. ISO (1997), ISO/IEC 14143-1:1997-Information technology-Software measurement Functional size measurement Definition of concepts, International Organization for Standardization, Geneva, Switzerland, 1997.

11. Kitchenham B.A., Travassos G.H.,Mayrhauser A.v., Niessink F., Schneidewind N.F. Singer J., Takada S., Vehvilainen R., Yang H. (1999), "Toward an ontology of software maintenance," Journal of Software Maintenance 1999; 11 (6):365—389.

12. Nguyen V., Huang L., Boehm B.W. (2010), "Analysis of Productivity Over Years", Technical Report, USC Center for Systems and Software Engineering.

13. Parikh G. and Zvegintzov N. (1983). The World of Software Maintenance, Tutorial on Software Maintenance, IEEE Computer Society Press, pp. 1-3.

14. Ramil J.F. (2003), "Continual Resource Estimation for Evolving Software," PhD Thes.is, University of London, Imperial College of Science, Technology and Medicine.

15. Reddy C.S., Raju K., (2009), "An Improved Fuzzy Approach for COCOMO's Effort Estimation using Gaussian Membership Function.

16. UKSMA (1998) Mkll Function Point Analysis Counting Practices Manual. United Kingdom Software Metrics Association. Version 1.3.1

17. Valerdi R. (2005), "The Constructive Systems Engineering Cost Model (COSYSMO)", PhD Thesis, The University of Southern California.