Design & Development of a Suitable Model to Validate New Estimation Approaches for Effective Performance

Safia Yasmeen, G. Manoj Someswar

Abstract: ***As COCOMO is a non-proprietary model, its details are available in the public domain, encouraging researchers and practitioners in the software engineering community to independently evaluate the model. There have been many extensions independently reported where machine learning techniques are used to generate effort models from the original COCOMO model. In our research work, we proposed a calibration method by transforming the model equation into a linear form and estimating the model parameters using standard linear regression techniques. This calibration method has been adopted by the COCOMO development team in their calibration work. COCOMO has also been a model used to validate new estimation approaches such as fuzzy logic and neural networks. The COCOMO development team continues to calibrate and extend the model using different calibration approaches on more augmented data sets and our research work is mainly based upon these approaches wherein we have evolved newer and better approaches over the existing approaches and gave a realistic outlook to the very purpose of achieving the best performance.***

Keywords: Calibration Technique, Maintenance Function Point (MFP), Maintenance Impact Ratio (MIR), Developed for Reusability (RUSE), Required Development Schedule (SCED), Required Software Reliability (RELY)

I. INTRODUCTION

In the last 20 years or so, a lot of initiations have been made by several researchers and this helped us in providing for an accurate and more reliable model to validate new estimation approaches for effective performance. Table 2-2 shows the best results obtained by main studies performed by our team. PRED (0.30) is the percentage of estimates that fall within 30% of the actuals. For example, PRED (0.30) = 52% indicates that the model in Clark [1998] produces estimates that are within 30% of the actuals, 52% of the time. Two types of results are often reported, fitting and cross-validation.[1] The fitting approach uses the same data set for both training and testing the model while the cross-validation approach uses different data sets: one for training and the other for testing. The accuracies reported from cross-validation better indicate the performance of the model because the main reason to build a model is to use it in estimating future projects.

Revised Version Manuscript Received on September 20, 2016.

Safia Yasmeen, Research Scholar, Mahatma Gandhi Kashi Vidyapith, Varanasi (Uttar Pradesh)-221002, India.

Dr. G. Manoj Someswar, Prof. Research Supervisor, Mahatma Gandhi Kashi Vidyapith, Varanasi (Uttar Pradesh)-221002, India.

Table 2. 2. COCOMO II Calibrations

Study	Fitting PRED (0.30)	Cross-validation PRED (0.30)
COCOMOII.1997 [Clark 1998]	52%	-
COCOMOII.2000 [Chulani 1999b]	75%	69%
COCOMOII.2003 [Yang and Chark 2003]	56%	-
Chen [2006]	-	76%
Nguyen [2008]	80%	75%

In COCOMOII.1997, the parameters are weighted averages of data-driven regression results and expert-judgment rating scales, in which the latter scales account for 90% of the weight. The COCOMOII.2000 calibration was based on the Bayesian analysis using a data set of 161 data points, and the COCOMOII.2003 calibration used the same Bayesian analysis, but it included 43 additional new data points. On the same data set as metrics measured at coarse granularity levels, modules (number of added, changed modules) and subsystems (number of added, changed), and number of changes to modules plus all changed modules. The models were calibrated and validated on the data sets collected from two case studies. In terms of MMRE and MdMRE, the best model achieved MMRE = 19.3%, MdMRE=14.0% and PRED (0.25)=44%. This best model seems to be based on coarse-grained metrics (subsystems), which is consistent with a prior finding by us in which coarse-grained metrics, e.g., the number of classes, were shown to estimate change effort more accurately than other finer-grained metrics. However, it is important to note that this best model did not generate the best (highest) PRED(0.25), indicating that the model evaluation and ensuing inferences are likely contingent upon the estimation accuracy indicators used.

In our research work, we described an improvised method and a supporting tool for dynamic calibration of the effort estimation model of renewal (reverse engineering and restoration) projects. The model accepts the change information gathered during the project execution and calibrates itself to better reflect dynamics, current and future trends of the project. At the beginning of the project, the model starts with its most common form calibrated from completed projects. During the project execution, the model may change its predictors and constants using the stepwise regression technique. The method uses fine-grained metrics obtained from the source code such as number of lines of source code. McCabe's cyclomatic complexity, Halstead's complexity and a number of modules were obtained after the renewal process.[2] We validated the method using both data from a legacy system and simulation. We found that

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

line-grained estimation and model dynamic recalibration are effective for improving the model accuracy and confirmed that the estimation model is process-dependent. A later empirical study further verifies these conclusions. Other studies have also reported some success in improving the prediction accuracies by recalibrating estimation models in the iterative development environment.

We proposed a model called Man Cost for estimating software maintenance cost by applying different sub-models for different types of maintenance tasks. We grouped maintenance tasks into four different types, hence, four sub-models:

- Error correction: costs of error correction for a release.
- Routine change: maintenance costs implementing routine change requests.
- Functional enhancement: costs of adding, modifying, deleting, improving software functionality. The model treats this type the same as new development in which the size can be measured in SLOC or Function Point, and the effort is estimated using adjusted size, complexity, quality, and other influence factors. -
  Technical renovation: costs of technical improvements, such as performance and optimization.[4]

Our research work suggests that these task types have different characteristics. Thus, each requires an appropriate estimation sub-model. Nonetheless, the adjusted size and productivity index are common measures used in these sub-models. The adjusted size was determined by taking into account the effects of complexity and quality factors.[5]

Although examples were given to explain the use of the sub-models to estimate the maintenance effort, there was no validation reported to evaluate the estimation performance of these sub-models.[6] We thus, also proposed several extensions to account for reengineering tasks and maintenance of web applications.

The wide acceptance of the FSM methods has attracted a number of studies to develop and improve maintenance estimation models using the FSM metrics.[7] Most of these studies focused on the cost of adaptive maintenance tasks that enhance the system by adding, modifying, and deleting existing functions. Having found that the FPA did not reflect the size of small changes well, we presented an extension to the FPA method by dividing the function complexity levels into finer intervals. This extension uses smaller size increments and respective weights to discriminate small changes that were found to be common in the maintenance environment.[8] We validated the model on the data obtained from a financial institution and demonstrated that a finer grained sizing technique better characterizes the size characteristics of small maintenance activities.

We also described a Maintenance Function Point (MFP) model to predict the effort required to implement non-corrective change requests. The model uses the same FPA procedure for enhancement to determine the FP count, but the Unadjusted FP count was adjusted by a multiplicative factor, namely Maintenance Impact Ratio (MIR), to account for the relative impact of a change.[9] The approaches were validated using the data set collected from a large financial information system, the best model producing relatively low prediction accuracies MMRE = 47% and PRED(0.25) = 28%. The result also shows that the size of the component to be changed has a higher impact on the effort than the size of the change. This result indicates that the maintainers might have spent time to investigate not only the functions affected by the change but also other functions related to the change.

Past research reported on the application of the COSMIC-FFP functional size measurement method to building effort estimation models for adaptive maintenance projects.[10] They described the use of the functional size measurement method in two field studies, one with the models built on 15 projects implementing functional enhancements to a software program for linguistic applications, the other with 19 maintenance projects of a real-time embedded software program.[11] The two field studies did not use the same set of metrics but they include three metrics, effort, Cfsu, and the level of difficulty of the project.[12] The authors showed that while project effort and functional size has a positive correlation, this correlation is strong enough to build good effort estimation models that use a single size measure. However, as they demonstrated, a more reliable estimation model can be derived by taking into account the contribution of other categorical factors, such as project difficulty.

II. TASK-LEVEL MODELS

The task-level model estimates the cost of implementing each maintenance task which comes in the form of error reports or change requests. This type of model deals with small effort estimates, usually ranging from a few hours to a month.[13]

We introduced a seven-step process and a tool called Soft Calc to estimate the size and costs required to implement maintenance tasks. The model uses various size measures, including SLOC (physical lines of code and statements), function points, object-points, and data points (the last two were originally proposed in the same paper). The size of the impact domain, the proportion of the affected software, was determined and then adjusted by complexity, quality, and project influence factors. The maintenance effort was computed using the adjusted size and a productivity index.

Rather than generating an exact effort estimate, it would be beneficial to classify maintenance change requests in terms of levels of difficulty or levels of effort required, and use this classification information to plan resources respectively. As an extension of our further research work, we proposed a modeling approach to building classification models for the maintenance effort of change requests. The modeling procedure involves four high-level steps, identifying predictable metrics, identifying significant predictable metrics, generating a classification function, and validating the model.[14] The range of each predictable variable and effort was divided into intervals, each being represented by a number called difficulty index. The effort range has five intervals (below one hour, between one hour and one day, between one day and one week, between one week and one month, above one month) which were indexed, from 1 to 5 respectively. To evaluate the approach,

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

we used a data set of 163 change requests from four different projects at Global Research Academy, Hyderabad, and Telangana, India. The approach produced the classification models achieving from 74% to 93% classification correctness.

Our modeling approach can be implemented to dynamically construct models according to specific environments. Organizations can build a general model that initially uses a set of predefined metrics, such as types of modification; the number of components added, modified, deleted; the number of lines of code added, modified, deleted. This general model is applied at the start of the maintenance phase, but it will then be then refined when data is sufficient. However, as our research work pointed out, it is difficult to determine the model's inputs correctly as they are not available until the change is implemented.

We presented a classification model that classifies the cost of rework in a library of reusable software components, i.e. Ada files. The model, which was constructed using the C4.5 mining algorithm developed by Quinlan in 1993 determines which component versions were associated with errors that require a high correction cost (more than 5 hours) or a low correction cost (no more than 5 hours).[15] Three internal product metrics, the number of function calls, the number of declaration statements, the number of exceptions, all were shown to be relevant predictors of the model. As these metrics can be collected from the component version to be corrected, the model can be a useful estimation tool. We then evaluated eleven different models to estimate the effort of individual maintenance tasks using regression, neural networks, and pattern recognition approaches

In the last approach, the Optimized Set Reduction (OSR) was used to select the most relevant subset of variables for the predictors of effort. All of the models use the maintenance task size, which is measured as the sum of added, updated, and deleted SLOC, as a main variable. Four other predictors were selected as they were significantly correlated with the maintenance productivity.[16] These are all indicator predictors: Cause (whether or not the task is corrective maintenance), Change (whether or not more than 50% of effort is expected to be spent on modifying of the existing code compared to inserting and deleting the code), Mode (whether or not more than 50% of effort is expected to be spent on development of new modules), Confidence (whether or not the maintainer has high confidence on resolving the maintenance task).

Other variables, type of language, maintainer experience, task priority, application age, and application size, were shown to have no significant correlation with the maintenance productivity. As our research work indicated, this result did not demonstrate that each of these variables has no influence on maintenance effort. There were possible joint effects of both investigated and non-investigated variables. For example, experienced maintainers wrote more compact code while being assigned more difficult tasks than inexperienced maintainers, and maintainers might be more experienced in large applications than in the small ones.[17] To validate the models, Jorgensen used a data set of 109 randomly selected maintenance tasks collected from different applications in the same organization. Of the eleven models built and compared, the best model seems to be the types of log linear regression and hybrid type based on pattern recognition and regression. The best model could generate effort estimates with MRE = 100% and PRED(.25) = 26% using a cross-validation approach used in previous work.[18] This performance is unsatisfactorily low. Considering low prediction accuracies achieved, we recommend that a formal model be used as supplementary to expert updated, and deleted SLOC, as a main variable. Four other predictors were selected as they were significantly correlated with the maintenance productivity. These are all indicator predictors: Cause (whether or not the task is corrective maintenance), Change (whether or not more than 50% of effort is expected to be spent on modifying of the existing code compared to inserting and deleting the code), Mode (whether or not more than 50% of effort is expected to be spent on development of new modules), Confidence (whether or not the maintainer has high confidence on resolving the maintenance task).

Other variables, type of language, maintainer experience, task priority, application age, and application size, were shown to have no significant correlation with the maintenance productivity. As our research work indicated, this result did not demonstrate that each of these variables has no influence on maintenance effort. There were possible joint effects of both investigated and non-investigated variables. For example, experienced maintainers wrote more compact code while being assigned more difficult tasks than inexperienced maintainers, and maintainers might be more experienced in large applications than in the small ones.[19] To validate the models, we used a data set of 109 randomly selected maintenance tasks collected from different applications in the same organization. Of the eleven models built and compared, the best model seems to be the types of log linear regression and hybrid type based on pattern recognition and regression. The best model could generate effort estimates with MRE = 100%
and PRED(.25) = 26% using a cross-validation approach. This performance is unsatisfactorily low.

The types of corrective maintenance tasks can improve the performance of the estimation model.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

Table 2-3: Maintenance Cost Estimation Models

Model/Study	Maintenance Task Type	Effort Estimated For	Modeling Approach	Key Input Metrics	Best Estimation Accuracies Reported[6]
COCOMO	Regular maintenance Major enhancement (Adaptation and reuse)	Maintenance period Adaptation and reuse project	Linear and Nonlinear Regression Bayesian Analysis	SLOC added and modified, SU, UNFM, DM, CM. IM	
KnowIedge Plan	Regular maintenance Major enhancement	Maintenance period Adaptation and reuse project	Arithmetic	SLOC or IFPUG'S FP of code added, reused, leveraged, prototype, base, changed, deleted, system base	
PRICE-S	Regular maintenance Major enhancement	Maintenance period Adaptation and reuse project	Arithmetic	SLOC, IFPUG's FP, POP, or UCCP of new, adapted, deleted, reused code.	
SEER-SEM	Corrective Adaptive Perfective	Maintenance period Adaptation and reuse project	Arithmetic	Maintenance rigor, Annual change rate, Years of maintenance, Effective size (measured in SLOC or IFPUG's FP)	
SLIM	Corrective Adaptive Perfective	Maintenance period	Heuristic	SLOC added and modified	
Basili 1996	Error correction and enhancement	Release	Simple linear regression	SLOC (sum of SLOC added, modified, deleted)	-
Basili 1997	Corrective	Component version	Classification (C4.5 algorithm)	Number of function calls, declaration statements, exceptions	76% classification correctness.
Niessink and van Vliet 1998	Adaptive (functional enhancement)	Individual chanjge request	Linear regression	IFPUG's FPA, Maintenance Impact Ratio (MIR)	MMRE=47% PRED(0.25)=28%
Abran 1995	Adaptive (functional enhancement)	A set of activities resulting a maintenance work product (Release)	Arithmetic	Extended IFPUG's FPA	
Abran 2002	Adaptive (functional enhancement)	Release (project)	Linear and nonlinear regression	COSMIC-FFP	MMRE=0.25 PRED(0.25)=53%

The selection of the best model is dependent on the performance indicators used. In this table, the most common performance indicators **MMRE and PRED (0.25)** are reported, and **MMRE** is used to indicate the bestmodel, the model that generates lowest **MMRE** values.

Table 2-3: Continued

Caivano 2001	Adaptive (renewal e.g.. reverse engineering and restoration)	Iteration	Linear regression Dynamic calibration	SLOC McCabe's complexity Halstead's complexity Number of modules	
Snced 2004	Various: Error correction Routine change Functional enhancement Technical renovation	Release	Productivity index COCOMOII	SLOC IFPUG's FP\Object-points [Sneed 1995] Number of errors Test coverage Complexity index Quality index Productivity index	
Ramil 2000.	All task types for	Release	Linear	Number of systems, modules	MMRE=19.3%

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

2003	evolving the software		regression		PRED(0.25)=44 % (Cross-validation)
Sneed 1995	Corrective Adaptive Perfective Preventive	Individual maintenance task	Productivity index (multiplied by adjusted size)	SLOC IFPUG's FP Object-points Data-points Complexity index Quality index Project influence factor	
Briand and Basili 1992a	Corrective Adaptive (and enhancive)	Individual change request	Classification	Modification type Source of error SLOC added, modified, deleted Components added, modified, deleted Objects (code or design or both)	74%-93% classification correctness
Jorgensen 1995	Corrective, adaptive, perfective, preventive	Individual maintenance task	Linear regression Neural networks Pattern recognition	SLOC (sum of SLOC added, modified, deleted)	MMRE=100% PRED(0.25)=26 % (Cross-validation)
Dc Lucia 2005	Corrective	Monthly period effort	Linear regression	Number of tasks SLOC	MMRE=32.25% PRED(0.25)=49. 32% (Cross-validation)

III. SUMMARY OF MAINTENANCE ESTIMATION MODELS

It is clear from the maintenance estimation models that estimating the cost required to develop and deliver a release receives much attention. Much attention given to the release estimation may indicate the widespread practice in software maintenance that the maintenance work is organized into a sequence of operational releases. Each release includes enhancements and changes that can be measured and used as a size input for the effort estimation model. Source code based metrics are the dominant size inputs, reflecting the fact that these metrics are the main metrics for effort estimation models in new development. Modified and added code are used in most of the models while some models use deleted code.

Although these studies report a certain degree of success in applying effort estimation models to the maintenance context, they exhibit several limitations. On the one hand, all of the models are context-specific with a few exceptions. They were built to address certain estimation needs of a single organization or several organizations performing maintenance work with specific processes, technologies, and people. Thus, this practical approach limits statistical inferences that can be made to other maintenance contexts that are different from where the studies were carried out. On the other hand, even with the models that are generic enough, there are a lack of empirical studies documenting and validating the application of the proposed models across maintenance contexts in multiple organizations with diverse processes, technologies, and people. Building generic models that can be recalibrated to specific organizations or even projects would be much needed in Maintenance estimation.

IV. 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 that the software components are reusable. In software maintenance, the maintain team usually adapts, reuses, or modifies the existing reusable components, and thus, $^{+}$his 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.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

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 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." 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. 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." [Boehm 2000b] 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:

$PM = A*Size^{E} * f\EM_i$ (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.

E M - 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.

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:

Log (PM) = fy + Pi \og{Size) + SF, \og(Size) + ...+ (Eq. 4-12)

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

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

approach such as ordinary least squares regression. This is a typical method that was used to calibrate the model coefficients and constants by 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. 4-11).

V. 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.

VI. DESCRIPTION OF THE EXPERIMENT WITH RESULTS

We recruited 1 senior and 23 computer-science graduate students who were participating in our directed research projects. The participation in the experiment was voluntary although we gave participants a small incentive by exempting participants from the final assignment. By the time the experiment was carried, all participants had been asked to compile and test the program as a part of their directed research work. However, according to our pre-experiment survey, their level of unfamiliarity with the program code (UNFM) varies from "Completely unfamiliar" to "Completely familiar". We rated UNFM as "Completely unfamiliar" if the participant had not read the code and as "Completely familiar" if the participant had read and understood the code, and modified some parts of the program prior to the experiment.

The performance of participants is affected by many factors such as programming skills, programming experience, and application knowledge. We assessed the expected performance of participants through pre-experiment surveys and review of participants' resumes. All participants claimed to have programming experience in either C/C++ or Java or both, and 22 participants already had working experience in the software industry. On average, participants claimed to have 3.7 years of programming experience and 1.9 years of working experience in the software industry.

We ranked participants by their expected performance based on their C/C++ programming experience, industry experience, and level of familiarity with the program. We then carefully assigned participants to three groups in a manner that the performance capability among the groups is balanced as much as possible. As a result, we had seven participants in the enhancive group, eight in the predictive group and nine in the corrective group.

Participants performed the maintenance tasks individually in two sessions in a software engineering lab. Two sessions had the total time limit of 7 hours, and participants were allowed to schedule their time to complete these sessions. If participants did not complete all tasks in the first session, they continued the second session on the same or a different day. Prior to the first session, participants were asked to complete a pre-experiment questionnaire on their understanding of the program and then were told how the experiment would be performed. Participants were given the original source code, a list of maintenance activities,, and a timesheet form. Participants were required to record time on paper for every activity performed to complete maintenance tasks. The time information includes start clock time, stop clock time, and interruption time measured in minute. Participants recorded their time for each of the following activities:

- Task comprehension includes reading, understanding task requirements, and asking for further clarification.
- Isolation involves locating and understanding code segments to be adapted. Editing code includes programming and debugging the affected code.
- Unit test involves performing tests on the affected code.

We focused on the context of software maintenance where the programmers perform quick fixes according to customer's maintenance requests. Upon receiving the maintenance request, the programmers validate the request and contact the submitter for clarifications if needed. They then investigate the program code to identify relevant code fragments, edit, and perform unit tests on the changes.

REFERENCES

1. 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
2. Albrecht A.J. (1979), "Measuring Application Development Productivity," Proc. IBM Applications Development Symp., SHARE-Guide, pp. 83-92.
3. 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
4. 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.
5. Basili V.R., (1990) "Viewing Maintenance as Reuse-Oriented Software Development," IEEE Software, vol. 7, no. 1, pp. 19-25, Jan.
6. 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.
7. 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
8. Boehm B.W. (1981), "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ, 1981.
9. Boehm B.W. (1988), "Understanding and Controlling Software Costs", IEEE Transactions on Software Engineering.
10. Boehm B.W., Royce W. (1989), "Ada CCCOMO and Ada Process Model," Proc. Fifth COCOMO User's Group Meeting, Nov.
11. Boehm B.W., Clark B., Horowitz E., Westland C, Madachy R., Selby

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

R. (1995), "Cost models for future software life cycle processes: COCOMO 2.0, Annals of Software Engineering 1, Dec, pp. 57-94.
12. Boehm B.W. (1999), "Managing Software Productivity and Reuse," Computer 32, Sept., pp.111-113
13. Boehm B.W., Abts C, Chulani S. (2000), "Software development cost estimation approaches: A survey," Annals of Software Engineering 10, pp. 177-205.
14. Boehm B.W., Horowitz E., Madachy R, Reifer D., Clark B.K., Steece B., Brown A.W., Chulani S., and Abts C. (2000), "Software Cost Estimation with COCOMO II," Prentice Hall.
15. Boehm B.W., Valerdi R. (2008), "Achievements and Challenges in Cocomo-Based Software Resource Estimation," IEEE Software, pp. 74-83, September/October
16. Bradley E., Gong G. (1983), "A leisurely look at the bootstrap, the jack-knife and cross-validation", American Statistician 37 (1), pp.836-848.
17. Briand L.C., Basili V., Thomas W.M. (1992), "A pattern recognition approach for software engineering analysis", IEEE Transactions on Software Engineering 18 (11)931-942.
18. Briand L.C. & Basili V.R. (1992) "A Classification Procedure for an Effective Management of Changes during the Software Maintenance Process", Proc. ICSM '92, Orlando, FL.
19. Briand L.C, El-Emam K., Maxwell K., Surmann D., and Wieczorek I., "An Assessment and Comparison of Common Cost Estimation Models," Proc. 21st International Cor ference on Software Engineering, pp. 313-322, 1999.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.