International Journal of Research
e-ISSN: 2348-6848 & p-ISSN 2348-795X Vol-5, Special Issue-11
International Conference on Multi-Disciplinary Research - 2017 held in February, 2018 in Hyderabad, Telangana State, India organised by GLOBAL RESEARCH ACADEMY - Scientific & Industrial Research Organisation (Autonomous), Hyderabad.

A Comprehensive Research Study on Participant's Maintenance Effort and Effort Adjustment Factor for Performance Appraisal of Computer Programmers

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

1. Research Scholar, Mahatma Gandhi Kashi Vidyapith, Varanasi, U.P., India

2. Research Supervisor, Mahatma Gandhi Kashi Vidyapith, Varanasi, U.P., India

Abstract:

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 redictive group, and nine in the corrective group.

***Keywords:** Participant's Maintenance Effort, Effort Adjustment Factor (EAF), Language and Tools Experience (LTEX), Platform Experience (PLEX), DELPHI SURVEY, Execution Time Constraint (TIME), Personnel Continuity (PCON)*

INTRODUCTION

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.

Obviously, the activities above do not include design modifications because small changes and enhancements hardly affect, the system design. Indeed, since we focus on the maintenance quick-fix, the maintenance request often does no, affect the existing design integration test activities are also no. included as the program is by itself the only component, and we perform acceptance testing independently to certify the completion of tasks.

Participants performed the maintenance tasks for the UCC program, an enhanced Version of the Code Count tool.[1] The program was a utility used to count and compare SLOC-related metrics such as statements, comments, directive statements, and data Legations of a source program. (This same program was also used to collect the sample data for calibrating the model proposed in this dissertation). UCC was written m C++ and |ad 5,188 logical SLOC in 20 classes.

The maintenance tasks were divided into three groups, enhancive, reductive, and Corrective, each being assigned to one respective participant group. These maintenance types fall into the business rules cluster, according to the topology proposed by us. There were five maintenance tasks for the enhancive group and six for the other groups.

The enhancive tasks require participants to add five new capabilities that allow the program to take an extra input parameter, check the validity of the input and notify users, count for and while statements, and display a progress indicator.[2] Since these capabilities are located in multiple classes and methods, participants had to locate the appropriate

code to add and possibly modify or delete the existing code. We expected that majority of code would be added for the enhancive tasks unless participants had enough time to replace the existing code with a better version of their own.

The reductive tasks ask for deleting six capabilities from the program. These capabilities involve handling an input parameter, counting blank lines, and generating a count summary for the output files. The reductive tasks emulate possible needs from customers who do not want to include certain' capabilities in the program because of redundancy, performance issues, platform adaptation, etc. Similar to the enhancive tasks, participants need to locate the appropriate code and delete lines of code, or possibly modify and add new code to meet the requirements.

The corrective tasks call for fixing six capabilities that were not working as expected. Each task is equivalent to a user request to fix a defect of the program. Similar to the enhancive and reductive tasks, corrective tasks handle input parameters, counting functionality, and output files. We designed these tasks in such a way that they required participants to mainly modify the existing lines of code.

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.

The first three charts in Figure 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. The task comprehension activity required the smallest proportions of effort. The corrective group spent the largest share of time for code isolation, twice as much as that of the enhancive group, while the reductive group spent much more time on unit test as compared with the other groups. That is, updating or deleting existing program capabilities requires a high proportion of effort for isolating the code while adding new program capabilities needs a large majority of effort for editing code.

The enhancive group spent 53% of total time on editing, twice as much as that spent by the other groups. At the same time, the corrective group needed 51% of the total time on program comprehension related activities including task comprehension and code isolation. Overall, these activities account for 42% of the total time. These results largely support the COCOMO IPs assumption in the size parameter software Understanding (SU) and the previous studies of effort distribution of

maintenance tasks which suggest that program comprehension requires up to 50% of maintainer's total effort.

Predicting Participant's Maintenance Effort

With the data obtained from the experiment, we built models to explain and predict time spent by each participant on his or her maintenance tasks. Previous studies have identified numerous factors that affect the cost of maintenance work.[3] These factors reflect the characteristics of the platform, program, product, and personnel of maintenance work. In the context of this experiment, personnel factors are most relevant. Other factors are relatively invariant, hence irrelevant, because participants performed the maintenance tasks in the same environment, same product, and same working set. Therefore, we examined the models that use only factors that are relevant to the context of this experiment.

Effort Adjustment Factor (EAF) is the product of the effort multipliers defined in the COCOMO II model, representing overall effects of the model's multiplicative factors on effort. In this experiment, we defined EAF as the multiplicative of programmer capability {PCAP), language and tools experience (LTEX), and platform experience (PLEX). We used the same rating values for these cost drivers that are defined in the COCOMO II Post-Architecture model. We rated PCAP, LTEX, PLEX values based on participant's GPA, experience, pre-test, and post test scores. The numeric values of these parameters are given in Appendix D. If the rating fell in between two defined rating levels, we divided the scale into finer intervals by using a linear extrapolation from the defined values of two adjacent rating levels. This technique allowed specifying more precise ratings for the cost drivers.

The estimates of the intercept (fio) in the models indicate the average overhead of the participant's maintenance tasks.[4] The overhead seems to come from non-coding activities such as task comprehension and unit test, and these activities do not result in any changes in source code. Model Mj has the highest overhead (110 minutes), which seems to compensate for the absence of the deleted SLOC in the model.

The coefficient of determination (R2) values suggest that 75% of the variability in the effort is predicted by the variables in M2 while only 50%, 55%, and 64% of that predicted by the variables in Mi, M3, and M4, respectively. It is interesting to note that both models M3 and M4, which did not include the deleted SLOC, generated higher R2 values than did model Mi. Moreover, the R values obtained by models M2 and M4 are higher than those of models M/ and M3 that use a single combined size metric.

The MMRE, PRED(0.3), and PRED(0.25) values indicate that M2 is the best performer, and it outperforms Mi, the worst performer, by a wide margin. Model M2 produced estimates with a lower error average (MMRE = 20%) than did Mi (MMRE = 33%). For model M2, 79% of the estimates (19 out of 24) have the MRE values of less than or equal to 30%. In other words, the model produced effort estimates that are within 30% of the actuals 79% of the time. Comparing the performance of M2 and M4, we can get that the deleted SLOC contributes to

improving the performance of M2 over M4 as these models have the same variables except the deleted SLOC. This result reinforces the rejection of Hypothesis 1.

LIMITATIONS OF THE EXPERIMENT

As the controlled experiment was performed using the subjects of student programmers in the lab setting, a number of limitations are exhibited. Differences in environment settings between the experiment and real software maintenance may limit the ability to generalize the conclusions of this experiment. First, professional maintainers may have more experience than our participants. However, as all of the participants, with the exception of the senior, were graduate students, and most of the participants including the senior had industry experience, the difference in experience is not a major concern. Second, professional maintainers may be thoroughly familiar with the program, e.g., they are the original programmers. The experiment may not be generalized for this case although many of our participants were generally familiar with the program.[5] Third, a real maintenance process may be different in several ways, such as more maintenance activities (e.g., design change and code inspection) and collaboration among programmers. In this case, the experiment can be generalized to four investigated maintenance activities that are performed by an individual programmer with no interaction or collaboration with other programmers

DELPHI SURVEY RESULTS

Expert-judgment estimation, as the name implies, is an estimation methodology that relies on the experts to produce project estimates based on their experience as opposed to using formal estimation methods. Expert-judgment estimation is useful in the case where information is not sufficient or a high level of uncertainty exists in the project ' being estimated. Of many expert-judgment techniques introduced, Wideband Delphi has been applied successfully in determining initial rating values for the COCOMO-like models, such as COCOMO 11.2000 and COSYSMO.[6]

In this study, the Delphi exercise was also employed to reach expert consensus regarding the initial rating scales of the maintenance effort model. The results are treated as priori-knowledge to be used in the Bayesian and other calibration techniques.

The Delphi form was distributed to a number of software maintainers, project managers, and researchers who have been involving in maintaining software projects and in software maintenance research. In the Delphi form, the definitions of parameters, pre-determined rating levels, and descriptions of each rating level were given. The form also includes the initial rating scales from COCOMO 11.2000. These initial rating scales were intended to provide participants information on the current experience about software development cost so that participants can draw similarities and differences between software development and maintenance to provide estimates for software maintenance. Although the survey was distributed to various groups of participants, it turned out that only participants who were familiar with COCOMO returned the form with their result.

The results of the Delphi survey are presented in Table 5-3 with the last two columns showing the productivity range and variances of productivity ranges for each of the cost drivers. The productivity range represents the maximum impact of the respective "x)St driver on effort. As an illustration, changing the ACAP rating from Low to High would require the additional effort of 77%.

Initially, the Delphi survey was planned to be carried out in two rounds. However, as shown in the last row of Table 5-3, the experts' estimates were converged even in the first round. Therefore, I decided not to run the second round. One possible explanation for this early convergence is the participants' familiarity with the COCOMO model and its cost drivers. In addition, the initial rating values of the cost drivers were provided, offering information for participants to compare with COCOMO II estimates.[7]

Table 5-2 shows the differences in the productivity ranges between the COCOMO 11.2000 model and the Delphi survey. The Difference column indicates an increase (if positive) or a decrease (if negative) in level of impact of the respective cost driver on effort. As can be seen in Table 5-3 and Table 5-2, a few cost drivers have their productivity ranges changed significantly. The Program Complexity (CPLX) still has the highest influence on effort, but its impact in software maintenance is less than in software development, indicating that the experts believe that having the legacy system will reduce the effort spent for maintaining the same system (although the complexity of the system remains the same). Other cost drivers Analyst Capability (ACAP), Application Experience (APEX), and Personnel Continuity (PCON), Execution Time Constraint (TIME), Main Storage Constraint (STOR), and Programmer Capability (PCAP).

These data attributes are grouped into three categories: project general information, size measures, and effort and cost drivers. One organization used its own data collection form, but its core data attributes were consistent with the definitions provided in our forms.

For the cost drivers, an actual rating that falls between two defined rating levels an be specified, allowing finer-grained increments in the rating scale to more closely describe the true value of the cost driver. The increment attribute can be specified to increase the base rating, and the numeric rating for the cost driver is a linear extrapolation from the base rating and the next defined value.[8]

CM is the percentage of code modified in the adapted modules. Thus, it is computed as the ratio between SLOC Adapted and SLOC Pre-Adapted, or CM = SLOC Adapted / SLOC Pre-Adapted.

In total, 86 releases that met the above criteria were collected. All 86 releases were completed and delivered in the years between 1997 and 2009. The application domains of these releases can be classified into data processing, military - ground, management of information system, utilities, web, and others. Of these data points, 64 came from an organization member of the center's Affiliates, 14 from a CMMI-level 5 organization in Vietnam, and 8 from a CMMI-level 3 organization in Thailand. The first organization has been an active COCOMO user and calibrated the model for their inter ml use. The

other organizations collected analyzed project size, effort, and other metrics as a part of their CMMI compliant processes. The organization in Vietnam granted me permission to interview project managers and team members to fill out the data collection forms. For the organization in Thailand, several interviews with the representative were carried out in order to validate the data points provided by the project teams. These granted accesses helped alleviate variations that may have been caused by consistency in understanding the data attributes.

The size metrics were collected by using code counting tools to compare differentials between two baselines of the source program (see Figure 5-2). These size metrics are based on the logical SLOC definition originally described in Park [1992] and later adopted into the definition checklist for logical SLOC counts in the COCOMO model. According to this checklist, one source statement is counted as one logical SLOC, and thus, blanks and comments are excluded. One organization reported using various code counting tools, and the other organizations used the Code Count tool10 for collecting the size metrics. Although using the same counting definition, variations in the results generated by these SLOC counting tools may exist[9]. This problem, which is caused by inconsistent interpretations of the counting definition, is a known limitation of the SLOC metrics [10]. Nonetheless, the SLOC counts among the releases of the same program are highly consistent as each program used the same SLOC counting tool.

Effort was collected in person-hour and converted into person-month using COCOMO's standard 152 person-hours per person-month, avoiding variations created by different definitions of person-month among the organizations. However, as discussed in [11], unrecorded overtime can cause variations in the actual effort reported. Another source of variations arrives from the subjective estimates of originates from adapting the pre-existing modules, almost a third for adding new modules, and only a small percentage (7.5%) for testing and integrating the reused modules.

The scatter plot on PM versus Equivalent KLSOC of the dataset shown in Figure 5-4 indicates that the dataset is skewed with far fewer large projects than small projects and the variability of PM is considerably higher in large projects. Additionally, there is one extreme project that has effort almost three times as much as that of the second largest project. These characteristics are often seen in software datasets. A typical approach to handling these datasets is to apply the logarithmic transformation on both effort and size metrics. The logarithmic transformation takes into account both linear and non-linear relationships, and it helps ensure linear relationships between log-transformed variables. The scatter plot in Figure 5-5 shows that the logarithmic transformation can appropriately resolve these issues that exist in the data set.

Outliers can distort the linear regression model, affecting the accuracy and stability of the model. Unfortunately, software data sets often contain outliers. This problem is caused by inconsistency and ambiguity in the definition of software terms (i.e., size and effort), imprecision in the data collection process, and the lack of standardized software processes. To handle possible outliers in software

data sets, several techniques have been used often, including building robust regression models, transforming the data, and identifying and eliminating outliers from the rest of the data set [12]. To some extent, all of these techniques were used in this study.

Table 5-8 summarizes the rating values of the 20 cost drivers calibrated by the Gayesian analysis using the 80 releases of the final data set. Undefined rating scales for the cost drivers at the corresponding levels are indicated by the grayed blank cells. It should be noted that the rating value for the Nominal rating of the effort multipliers is 1.0, which implies that the nominal estimated effort is not adjusted by the effort multipliers.

Hypothesis 2 states that the productivity ranges of the cost drivers in the COMO II model for maintenance are different form those of COCOMO 11.2000. Different approaches used to calibrate the COCOMO II model for maintenance result in different sets of productivity ranges. For testing this hypothesis, the productivity ranges calibrated through the Bayesian analysis were used to compare with those of COCOMO 11.2000. This comparison is valid since the Bayesian analysis was also applied to calibrating COCOMO 11.2000's productivity ranges.

Table 5-7 presents productivity ranges of the COCOMO II maintenance calibrated y the Bayesian approach and COCOM 11.2000 and their differences between the two models. For the effort multipliers, the productivity range is the ratio between the largest and the smallest rating value. The productivity ranges of the scale factors are for 100 EKSLOC projects and computed as Pl.SFj = IQQQ" ' w^ere ^ an<^ SFimax are the constant B and the maximum rating value of the scale factor i.

Table 1: Estimation Accuracies of Constrained Approaches

Approach	PRED(0.30)	PRED(0.25)	PRED(0.50)	MMRE	MdMRE
CMRE	60%	56%	76%	37%	23%
CMSE	51%	43%	79%	39%	30%
CMAE	58%	54%	'71%	42%	23%

	PRED(0.30)	PRED(0.25)	PRED(0.50)	MMRE	MdMRE
CMRE	60%	56%	76%	37%	23%
CMSE	41%	35%	69%	49%	35%
CMAE	51%	46%	67%	52%	28%

Table 2: Estimation Accuracies of Constrained Approaches

Approach	PRED(0.30)	PRED(0.25)	PRED(0.50)	MMRE	MdMRE
CMRE	60%	56%	76%	37%	23%
CMSE	51%	43%	79%	39%	30%
CMAE	58%	54%	71%	42%	23%

Table 3: Estimation Accuracies of Constrained Approaches using LOOC Cross-validation

As illustrated in Table 1, which shows the productivity ranges generated by the CMRE approach, eight cost drivers, including all 5 scale factors but FLEX and 4 effort multipliers (PVOL, DATA, TIME, and RELY), were pruned from the model, leaving 12 most **relevant** cost drivers in the model. The **Storage Constraint** (STOR) cost driver appears to be the most influential driver with the productivity range of 2.81, and the personnel factors except LTEX are among the least influential. This result appears to be counterintuitive since it contradicts the Delphi results which indicate that personnel factors have the most significant impacts on effort and that STOR is ***1*** tot considered to be highly influential. It should be noted that, unlike the Bayesian approach, which adjusts the data-driven estimates with the experts' estimates, the constrained regression techniques rely heavily on the sample data to generate estimates. Thus, like multiple 'near regression, it suffers the correlation between cost drivers and the lack of dispersion the sample data set..

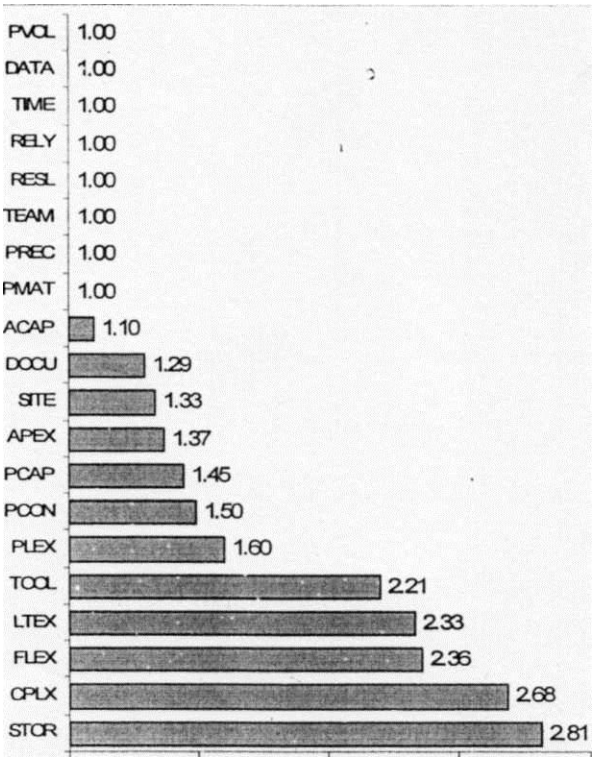

Figure 2: Productivity Ranges Generated by CMRE

The estimation accuracies produced by three constrained regression approaches are listed in Table. Considering the PRED(0.30), PRED(0.25), and MMRE metrics, it is clear that CMRE and CMAE outperform both COCOMO 11.2000 and the Bayesian calibrated model, improving PRED(OJO) from 38% produced by COCOMO 11.2000 to 60%, a 58% improvement. It is important to note that CMSE's PRED(0.30) and PRED(0.25) values are as low as those of the Bayesian calibrated model. A possible reason this similarity is that both CMSE and the Bayesian calibrated model optimize the same quantity, the sum of square errors. This quantity may overlook high square errors in small projects to favor high square errors in large projects, resulting in high MRE errors in the estimates of these small projects. The LOOC cross-validation results in Table 5 further confirm the improvement in the performance of the constrained models over the Bayesian calibrated model. This observation is consistent with the results reported in which the performance of these approaches were compared with Lasso, Ridge, Stepwise, and OLS regression techniques using other COCOMO data sets.

REDUCED PARAMETER MODELS

Strong correlations and the lack of dispersion in cost drivers can negatively affect the stability of the estimation model that is built on the assumptions of the multiple linear regression. The analysis of the correlation and dispersion properties of the data set may suggest possible redundant drivers that contribute little to improving the accuracy of the estimation model. Similar to the previous COCOMO studies, this study also investigated a reduced model with a smaller set of cost drivers by aggregating cost drivers that are highly correlated or lack dispersion. Two cost drivers are deemed to be aggregated if their correlation coefficient is 0.65 or greater.

Table 3 shows the correlation matrix of highly correlated drivers, all belonging to the personnel factors. ACAP and PCAP were aggregated into Personnel Capability (PERS), and APEX, LTEX, PLEX into Personnel Experience (PREX). It turns out that these aggregations are the same as those of the COCOMO Early Design model, except that PCON was not aggregated into PREX.

Table 3: Correlation Matrix for Highly Correlated Cost Drivers

ACAP	PCAP	APEX	LTEX	PLEX
PCAP	**0,74**	1.00		
APEX	0.44	0.39	1.00	
TEX	0.58	0.53	*0J2*	1.00
LEX	0:49	03	058	**75**

Table 3 and Table 4 show that the *Execution Time Constraint* (TIME) and *Main Storage Constraint* (STOR) ratings are positively skewed with 82 and **76** data oints out of the 86 data points rated Nominal,

respectively. Moreover, while the range of defined ratings for TIME and STOR is from Nominal to Very High, no data point was rated High and Very High for TIME and Very High for STOR. This lack of dispersion in TIME and STOR can result in high variances in the coefficients of these drivers as the impact of the non-Nominal ratings cannot be determined due to the lack of these ratings. An explanation for this lack of dispersion is that new technologies and advancements in the storage and processing facilities, making TIME and STOR insignificant for the systems that are less constrained by the limitations of the storage and processing facilities. And all of the projects in the data set are non-critical and non-embedded systems, which are typically not dependent on the storage and processing constraints. Thus, the TIME and STOR cost drivers were eliminated from the reduced model. In all, seven cost drivers were removed, and two new were added, resulting in the red; ced model with 15 cost drivers, as compared to the full model with 20 cost

drivers, gayesian approach appears to have a smaller MMRE value than that of the respective full model.

Local Calibration

A series of stratifications by organization and by program were performed using four different approaches:

- **Productivity index.** The project effort estimate is determined by dividing project size by productivity index, i.e., **Effort = Size I Productivity Index,** where the productivity index is the average productivity of past projects or an industry census. It is the simplest model to estimate effort given that the productivity index is known. This model is often referred to as the baseline, and a more sophisticated model is only useful if it outperforms the baseline model.
- **Simple linear regression between effort and size.** The simple linear regression is used to derive the model whose response and predictor **are** the logarithmic transformations of PM and **Equivalent KSLOC,** respectively.
- **Bayesian analysis.** The full model with all the drivers that were calibrated using the Bayesian analysis was used as a basis to fit into local data sets and compute the constants **A** and **B** in the COCOMO effort estimation model (Eq. 4-11). The resulting local models differ from each other only in the constants **A** and **B.** The full model, instead of the reduced model, was used **since** it allows organizations to select from the full model the most suitable cost drivers for their processes.
- **Constrained regression with CMRE.** This local calibration approach used the set of 12 cost drivers and the constants *A* and *B* obtained from the constrained regression with CMRE. The constants *A* and *B* were further calibrated into local data sets based on organization and program.

Using the approaches above, the following eight local calibrations were investigated:

CI. The productivity index of each organization was computed and used to estimate the projects within the same organization.

C2. The simple linear regression was applied for each individual organization.

C3. Stratification by organization using the Bayesian approach.

C4. Stratification by organization using the constrained approach CMRE.

C5. The productivity index of the previous releases in each program was used to determine the effort of the current release in the same program.

C6. The simple linear regression was applied for each individual program that has five or more releases.

C7. Stratification by program using the Bayesian approach.

C8. Stratification by program using the constrained approach CMRE.

International Journal of Research
e-ISSN: 2348-6848 & p-ISSN 2348-795X Vol-5, Special Issue-11
International Conference on Multi-Disciplinary Research - 2017 held in February, 2018 in Hyderabad, Telangana State, India organised by GLOBAL RESEARCH ACADEMY - Scientific & Industrial Research Organisation (Autonomous), Hyderabad.

Table 4: Stratification by Organization on 45 Releases

Calibration	#Releases	PRED(0.30)	PRED(0.25)	MMRE	MdMRE
CI	45	40%	40%	44%	37%
C2	45	31%	27%	53%	40%
C3	45	**63%**	**55%**	**40%**	**24%**
C4	45	**60%**	**58%**	**34%**	**22%**

It can be seen from Table 3 and Table 4 that the calibrations based on the gayesian and CMRE models outperform both productivity index and simple linear regression approaches. C3 and C4, whose performances are comparable, are more approachable than CI and C2 when stratification by organization is concerned. Similarly, C7 and C8 outperform C5 and C6 when calibrating for each individual program. It is clear at the effects of the cost drivers in the Bayesian and CMRE models positively contribute to improving the performance of the models. Even in the same program, the ost drivers' ratings change from one release to another. For example, if the attrition rate low, the programmer is more familiar with the system, and is more experienced with e languages and tools used in the program. If the attrition rate is high, on the other hand, the overall programmer capability and experience could be lower than the previous release. Due to these effects, the productivity of the releases of the same program does iot remain the same.

The program-based calibrations appear to generate better estimates than the organization-based calibrations. This finding is further confirmed when the local models t CI, C2, C3, and C4 were tested on the same data set of 45 releases used in the program-based calibrations. One explanation for the superiority of the program-based calibrations is that possible variations in application domains, technologies, and software processes were minimized when considering only releases in the same program. Another explanation is that the variations in the data collection process are reduced as the same code counting tool was used and the same data collector rated the cost drivers for all releases in the same program.

The best performers are the local models that were calibrated using the Bayesian and MRE models on each individual program. They could produce estimates within 30% of the actuals 80% of the time.

This result is comparable to the COOCOMO 11.2000 model when it was stratified by organization.

Comparing the accuracies in Tables 5-9, 5-12, 5-13, 5-16, 5-17, and, 5-18 one can see that the generic models calibrated using the Bayesian and CMRE approaches can be further improved by calibrating them into each individual organization and program. Moreover, these generic models are more favorable than the productivity index and the simple linear regression in the stratification by organization, indicating that the generic models may be useful in the absence of sufficient data for local calibration. This finding confirms the previous COCOMO studies and other studies which suggest that local calibration improves the performance of the software estimation model.

Hypothesis 4 states that the COCOMO II model for maintenance outperforms the simple linear regression and the productivity index method. As shown above, this hypothesis is supported.

Conclusion

The data set of 80 releases from 3 organizations was used to validate the proposed extensions to the size and effort estimation model. The effort estimation model was calibrated to the data set using a number of techniques including the linear regression, Bayesian, and constrained regression. Local calibrations to organization and program were also performed and compared with the less sophisticated approaches, the productivity index and the simple linear regression. The best model, which was calibrated using the releases of each individual program, can produce estimates with PRED(0.30) = 80% and MMRE = 0.22, outperforming the less sophisticated but commonly used productivity index and simple linear regression.

REFERENCES

1. Nguyen V., Deeds-Rubin S., Tan T., Boehm B.W. (2007), "A SLOC Counting Standard," The 22nd International Annual Forum on COCOMO and Systems/Software Cost Modeling. DOI = http://csse.usc.edu/csse/TECHRPTS/2007/usc-csse-2007-737/usc-csse-2007-737.pdf

2. Nguyen V., Steece B., Boehm B.W. (2008), "A constrained regression technique for COCOMO calibration", Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM), pp 213-222

3. Nguyen V., Boehm B.W., Danphitsanuphan P. (2009), "Assessing and Estimating Corrective, Enhancive, and Reductive Maintenance Tasks: A Controlled Experiment." Proceedings of 16th Asia-Pacific Software Engineering Conference (APSEC 2009), Dec.

4. Nguyen V., Boehm B.W., Danphitsanuphan P. (2010), "A Controlled Experiment in

Assessing and Estimating Software Maintenance Tasks", APSEC Special Issue, Information and Software Technology Journal, 2010.

5. Nguyen V., Huang L., Boehm B.W. (2010), "Analysis of Productivity Over Years", Technical

Report, USC Center for Systems and Software Engineering.

6. Niessink F., van Vliet H. (1998), "Two case study in measuring maintenance effort", Proceedings of International Conference on Software Maintenance, Bethesda, MD, USA, pp. 76-85.

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

8. Park R.E. (1992), "Software Size Measurement: A Framework for Counting Source Statements," CMU/SEI-92-TR-11, Sept.

9. Pew R. W. and Mavor A. S. (2007), "Human-System Integration in the System Development Process: A New Look", National Academy Press

10. Price-S (2009), TruePlanning User Manual, PRICE Systems, www.pricesystems.com

11. Putnam L. & Myers W. (1992), "Measures for Excellence: Reliable Software on Time, within Budget," Prentice H. 11 PTR.

12. Quinlan J.R. (1993), "C4.5: Prograns for Machine Learning," Morgan Kaufmann Publishers, Sao Mateo, CA.