Assessing and Estimating Corrective, Enhancive, and Reductive Maintenance Tasks: A Controlled Experiment

Vu Nguyen, Barry Boehm
Computer Science Department
University of Southern California
Los Angeles, USA
nguyenvu@usc.edu, boehm@usc.edu

Phongphan Danphitsanuphan
Computer Science Department
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
phongphand@kmutnb.ac.th

Abstract— **This paper describes a controlled experiment of student programmers performing maintenance tasks on a C++ program. The goal of the study is to assess the maintenance size, effort, and effort distribution of three different maintenance types and to describe estimation models to predict the programmer's effort on maintenance tasks. The results of our study suggest that corrective maintenance is much less productive than enhancive and reductive maintenance. Our study also confirms the previous results which conclude that corrective and reductive maintenance requires large proportions of effort on program comprehension activity. Moreover, the best effort model we obtained from fitting the experiment data can estimate the time of 79% of the programmers with the error of 30% or less.**

Keywords— software maintenance; software estimation; maintenance experiment; COCOMO; maintenance size

I. INTRODUCTION

Software maintenance is crucial to ensuring useful lifetime of software systems. According to previous studies [1][2][3], the majority of software related work in organizations is devoted to maintaining the existing software systems rather than building new ones. Despite advances in programming languages and software tools that have changed the nature of software maintenance, programmers still spend a significant amount of effort to work with source code directly and manually. Thus, it is still an important challenge in software engineering community to assess maintenance cost factors and develop techniques that allow programmers to accurately estimate their maintenance.

A typical approach to building estimation models is to determine what factors and how much they affect the effort at different levels and then use these factors as the input parameters in the models. For software maintenance, the modeling process is even more challenging. The maintenance effort is affected by a large number of factors such as size and types of maintenance work, personnel capabilities, the level of programmer's familiarity with the system being maintained, processes and standards in use, complexity, technologies, the quality of existing source code and its supporting documentation [4][5].

There has been tremendous effort in software engineering community to study cost-driven factors and the amount of impact they have on maintenance effort [6][7]. A number of models have been proposed and applied in practice such as [4][8][9]. Although maintenance size measured in source lines of code (LOC) is the most widely used factor in these models, there is a lack of agreement on what to include in the LOC metric. While some models determine the metric by summing the number of LOC added, modified, and deleted [9][10], others such as [4] use only LOC that is added and modified.

In this paper, we describe a controlled experiment of student programmers performing maintenance tasks on a small C++ program. The purpose of the study was to assess the size and effort implications and labor distribution of three different maintenance types and to describe estimation models to predict the programmer's effort on maintenance tasks. We focus the study on enhancive, corrective, and reductive maintenance types according to a maintenance topology proposed by Chapin et al [11]. We chose to study these types because they are the ones that change the business rules of the system by adding, modifying, and deleting the source code. The results of our study suggest that the corrective maintenance is less productive than enhancive and reductive maintenance. This result is consistent with the conclusion from previous studies [9][12]. The results also confirm previous studies of effort distribution of maintenance tasks: program comprehension requires as much as 50% of maintainer's total effort.

The rest of the paper is organized as follows. Section II provides a method for calculating the equivalent LOC in maintenance programs. The experiment design and results are discussed in Section III and IV. Section V describes models to estimate programmers' effort on maintenance tasks. Section VI discusses various threats to the validity of the research results; and the conclusions are given in Section VII.

II. CALCULATING EQUIVALENT LOC

In software maintenance, the programmer works on the source code of the existing system. The maintenance work is constrained by the existing architecture, design, implementation, and technologies used. This takes maintainers extra time to comprehend, test, and integrate the maintained pieces of code. Thus, an acceptable estimation model should reflect this affect in its either size or effort estimates.

In this experiment, we adapt the COCOMO II reuse model to determine the equivalent LOC of the maintenance tasks. The model involves determining the amount of software to be adapted, the percentage of design modified (DM), the percentage of code modified (CM), the percentage of integration and testing (IM), the degree of Assessment and Assimilation (AA), understandability of the existing software (SU), and the programmer's unfamiliarity with the software (UNFM). The last two parameters directly account for the programmer's effort to comprehend the existing system.

The equivalent SLOC formula is defined as

$$EquivalentSLOC = TRCF \times AAM \quad (1)$$

$$AAF = \frac{S}{TRCF}$$

$$AAM = \begin{cases} AAF(1 + [1 - (1 - AAF)^2] \times SU \times UNFM, \text{ for } AAF \le 1 \\ AAF + \frac{SU \times UNFM}{100}, \text{ for } AAF > 1 \end{cases}$$

Where,

- *TRCF* = the total LOC of task-relevant code fragments, i.e., the portion of the program that the maintainers have to understand to perform their maintenance tasks.
- *S* = the size in LOC.
- *SU* = the software understandability. SU is measured in percentage ranging from 10% to 50%.
- *UNFM* = the level of programmer unfamiliarity with the program. The *UNFM* rating scale ranges from 0.0 to 1.0 or from "Completely familiar" to "Completely unfamiliar".

LOC is the measure of logical source statements (i.e., logical LOC) according to COCOMO II's LOC definition checklist given in [4] and further detailed in [13].

Ko et al. studied maintenance activities performed by students, finding that the programmers collected working sets of task-relevant code fragments, navigated dependencies, and editing the code within these fragments to complete the required tasks [14]. This *as-needed strategy* [15] does not require the maintainer to understand code segments that are not relevant to the task.

The equation (*1*) reflects this strategy by including task-relevant code fragments other than the whole adapted program. The task-relevant code fragments are functions and blocks of code that are affected by the changes.

III. Description of the Experiment

A. Hypotheses

According to Boehm [16], programmer's maintenance activities consist of understanding maintenance task requirements, code comprehension, code modification, and unit testing. Although the last two activities deal with source code directly, empirical studies have shown high correlations between the overall maintenance effort and total LOC added, modified, and deleted (e.g., [9][17]). With the same settings as above, we hypothesize that these activities have comparable distributions of programmer's effort regardless of what types of changes are made. Indeed, with the same cost factors [4] such as program complexity, project and personnel attributes, the productivity of enhancive tasks is expected to have no difference with that of corrective and reductive maintenance. Thus, we have the following hypotheses:

Hypothesis 1: There is no difference in the productivity among enhancive, corrective, and reductive maintenance.

Hypothesis 2: There is no difference in the division of effort across maintenance activities.

B. The Participants and Groups

We recruited 1 senior and 23 graduate computer-science 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 test, their level of unfamiliarity with the program code (*UNFM*) varies from "Completely unfamiliar" to "Completely familiar" as 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 source code, and modified some parts the program prior to the experiment.

The performance of participants is affected by many factors such as programming skills, programming experience, and application knowledge [4][18]. We assessed the expected performance of participants through a pre-experiment survey 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 (±2) years of programming experience and 1.9 (±1.7) years of experience in the software industry.

We ranked the 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 each group in a manner that the performance capability among groups is balanced as much as possible. As a result, we had seven participants in the enhancive group, eight in the reductive group, and nine in the corrective group. We will further discuss in Section VI potential threats related to the group assignments, which may result in validity concerns of the results.

C. Procedure and Environment

Participants performed maintenance tasks individually in two sessions in a software engineering lab. Two sessions had total time limit of 7 hours, and participants were allowed to choose sessions relevant for their schedule. 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 to complete maintenance tasks; time information includes start clock time, stop clock time, and interruption time measured in minute.

Participants used Visual Studio 2005 on Windows XP. The program's user manual was provided to help participants setup the working environment and compile the program.

Prior to completing the assignments, participants were given prepared acceptance test cases and were told to run these test cases to certify their updated program. These test cases covered new, deleted, and affected capabilities of the program. Participants were also told to record all defects found during the acceptance test and not to fix or investigate these defects.

D. The UCC Program

The UCC was a program that allowed users to count LOC-related metrics such as statements, comments, directive statements, and data declarations of a source program. It also allowed users to compare the differentials between two versions of a source program and determine the number of LOC added, modified, and deleted. The program was developed and distributed by the USC Center for Systems and Software Engineering. The UCC program had three main modules (1) read input parameters and parse source code (2) analyze and count source code (3) produce results to output files. The UCC program had 5188 logical LOC and consisted of 20 C++ classes.

E. Maintenance Tasks

The maintenance tasks were divided into three groups, enhancive, reductive, and corrective, each being assigned to one participant group. These maintenance types fall into the *business rules* cluster, according to the topology proposed by Chapin et al. [11]. 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. 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 some capabilities in the program because of redundancy, performance issues, platform adaptation, etc. Similar to the enhancive tasks, the 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 code.

F. Metrics

The independent variable was the type of maintenance, consisting of enhancive, corrective, and reductive. The dependent variables were programmer's effort and size of change. Programmer's effort was defined as the total time the programmer spent to work on the maintenance task excluding interruption time; size of change was measured by three components, LOC added, modified, and deleted.

G. Maintenance Activities

We focus on the context of software maintenance where the programmer performs quick fixes according to customer's maintenance requests [19]. 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 [9][14]. In the experiment, we grouped this scenario into four maintenance 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.

Obviously, these activities do not include design modification because small changes and enhancements hardly affect the system design. Indeed, since we focus on the maintenance quick-fix, the maintenance request often does not affect the existing design.

IV. RESULTS

In this section, we provide the results of our experiment, the analysis and interpretation of the results. We use one-sided Mann-Whitney U Test with the typical 0.05 level of significance to test the statistically significant difference between the two sets of values. We also perform Kruskal-Wallis test to validate the differences among the groups.

A. Data Collection

Data was collected from three different sources including surveys, timesheet, and the source code's changes. From the surveys, we determined participants' programming skills,

industry experience, and level of unfamiliarity with the program.

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 which totaled 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 a real-world situation the acceptance test is usually performed by customers or an independent team, and their effort is often not recorded as the time spent by the maintenance team.

Changes size was collected in terms of the number of LOC added, modified, and deleted by comparing the original version with the modified versions. These LOC values were then adjusted using the formula (1) to obtained equivalent LOC. We measured the LOC of task-relevant code fragments (*TRCF*) by summing the size of the affected methods. We counted the LOC metric following the logical LOC definition given in the COCOMO II model. As a LOC is corresponding to one logical source statement, one LOC modified can be easily distinguished from a combination of one added and one deleted.

B. Task Completion

We did not expect participants to complete all the tasks, given their capability and availability. Because the tasks were independent, we were able to identify easily which source code changes are associated with each task. Six participants spent time on incomplete tasks with a total of 849 minutes or 18% of total time. On average, the enhancive group spent most time (26%) on incomplete tasks compared with 16% and 12% by the corrective and reductive groups, respectively.

Thereafter, we exclude time and size associated with incomplete tasks because the time spent on these tasks did not actually produce any result to meet the task requirements.

C. Distribution of Effort

Figure 1 shows the distributions of effort spent by participants on the enhancive, reductive, and corrective tasks.

Unexpectedly, proportions of effort spent on the maintenance activities vary vastly among the three groups. More than 50% of the time spent by the enhancive group is on editing, twice as much as that by the other groups. It is possible to infer that because of high proportion of time spent on coding, the participants in the enhancive group produced more LOC than did the other groups.

The corrective group spent a 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 much effort for editing code.

The maintainers on the enhancive group spent less time on the isolation activity but more time on writing code while the maintainers on the corrective group did the opposite, resulting in the total percentages of coding and code isolation activity of the two group almost the same (72% and 73%), respectively. The Kruskal-Wallis rank-sum tests confirm the differences in the percentage distributions of editing code (p=0.0095) and code isolation (p=0.0013) among these groups. The Hypothesis 2 is therefore rejected.

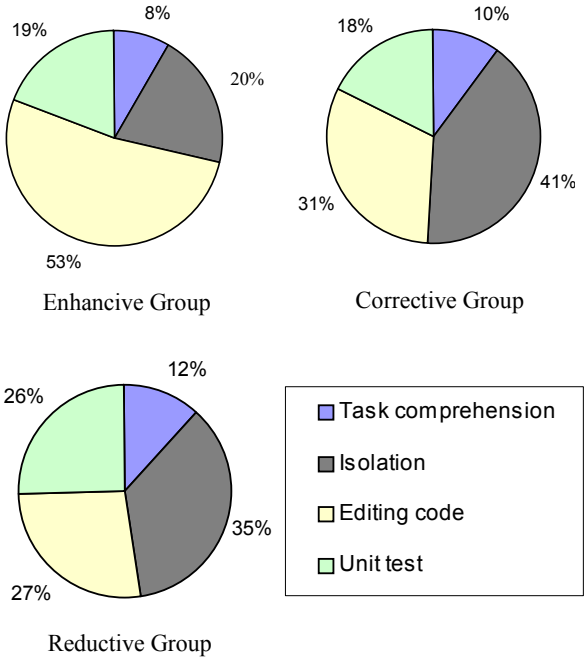

Figure 1. Effort distribution

D. Productivity

Figure 2 shows that, on average, the enhancive group produced almost 1.5 times as many LOC as did the reductive group and almost four times as many LOC as did the corrective group. Participants in the enhancive group focused on adding new, the reductive group on deleting existing, and the corrective group on modifying LOC. This pattern was dictated by our task design: the enhancive tasks require participants to mainly add new capabilities which result in new code, the corrective tasks require modifying existing code, and the reductive tasks require deleting existing code. For example, participants in the reductive group modified 20% and deleted the rest 80% of the total affected LOC.

The box plots shown in Figure 3 provide 1st quartile, median, 3rd quartile, and outliers of the productivity for three groups. The productivity is defined as the total number of equivalent added, modified, and deleted LOC divided by effort.

One participant had much higher productivity than any other participants. A closer look at this data point reveals that this participant had 8 years of industry experience and was working as a full-time software engineer at the time of experiment.

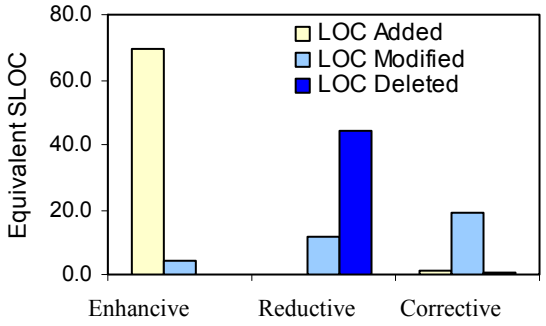

Figure 2. Average equivalent LOC added, modified and deleted in the three groups

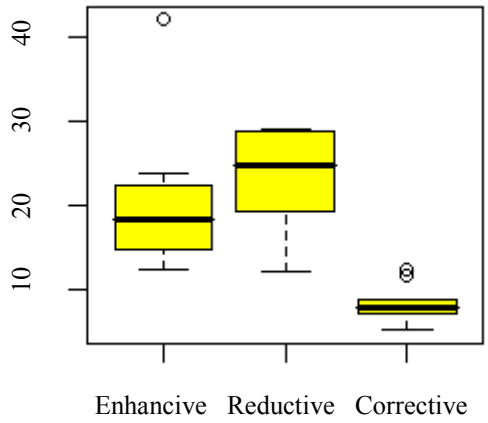

Figure 3. Participants Productivity

As indicated in the box plots, the productivity of the corrective group is much lower than that of the other groups. On average, for each hour participants in the corrective group produced 8 (± 1.7) LOC, which is approximately 2.5 times less than did the reductive and enhancive groups produce, 20 (± 8.4) and 21 (± 11.3) respectively. One-sided Mann-Whitney U test confirms these productivity differences (p = 0.001 for the difference between the enhancive and corrective groups and p = 0.0002 between the reducive and corrective groups); and there is a lack of statistical evidence to indicate the productivity difference between enhancive and reductive groups (p = 0.45). Kruskal-Wallis rank-sum test also indicates statistically significant difference in the productivity among these groups (p = 0.0004). The Hypothesis 1 is therefore rejected.

V. Explanatory Maintenance Effort Models

Understanding the factors that affect maintenance cost and predicting future cost is one of the best interests of software engineering practitioners. Reliable estimates enable practitioners to make informed decisions and ensure the success of software maintenance projects. With the data obtained from the experiment, we are interested in deriving models to explain and predict time of each participant on the maintenance tasks.

A. Models

Previous studies have identified numerous factors that affect the cost of maintenance work, including platform, program complexity, product, and personnel [4][18]. In the context of this experiment, personnel factors are most relevant. Other factors are relatively invariant because participants performed the maintenance tasks in the same environment, same product, and same working set.

Effort Adjustment Factor (EAF) is the product of effort multipliers of the COCOMO II model, representing the overall affect of the multiplicative factors on the effort. In this experiment, we define *EAF* as the multiplicative of programmer capability (*PCAP*), language 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*, *LEXT*, *PLEX* values based on participant's GPA, experience, pre-test and post test scores.

We will investigate the following models

$$\boldsymbol{M_1}: E = \beta_0 + \beta_1 * S * EAF \quad (2)$$

$$\boldsymbol{M_2}: E = \beta_0 + (\beta_1 * Add + \beta_2 * Mod + \beta_3 * Del) * EAF \quad (3)$$

E is the total time that a participant spends on completed maintenance tasks. *Add*, *Mod*, and *Del* represent the number of LOC added, modified, and deleted by the participant for all completed maintenance tasks, respectively. S is the total equivalent LOC that was added, modified, and deleted by the participant, that is, $S = Add + Mod + Del$. It is worthy noting that the difference between M_1 and M_2 is on the estimates of coefficients β_1, β_2, and β_3. M_1 and M_2 are identical if the estimates of coefficients β_1, β_2, and β_3 in M_2 receive the same value. This difference is subtle but significant because M_2 takes into account the impact of each type of LOC changes. In the next two subsections, we will estimate the coefficients of the models using the experiment data and evaluate the performance as well as the structural difference between them.

B. Model Performance Measures

We use two widely-used model performance measures MMRE and PRED as criteria to evaluate the accuracy of the models. These metrics are derived from the basic magnitude of the relative error MRE, which is defined as,

$$MRE_i = \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (4)$$

Where y_i and $\hat{y}_i$ are the actual and the estimate of the ith observation, respectively. Because y_i is log-transformed, we calculate the MRE_i using

$$MRE_i = \left| 1 - e^{\hat{y}_i - y_i} \right| \quad (5)$$

The mean of MRE of N estimates is defined as

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \quad (6)$$

PRED(l) is defined as the percentage of estimates where MRE is not *greater* than l, that is PRED(l) = k/N, where k is the number of estimates with MRE values falling within l. We chose to report MMRE, PRED(0.25), and PRED(0.3) values because they have been the most widely-used measures for evaluating the performance of the software estimation model [20][21][22].

C. Results

We collected the total of 24 data points, each having total equivalent LOC (S), LOC added (*Add*), modified (*Mod*), deleted (*Del*), actual effort (E), and effort adjustment factor (*EAF*). Fitting the 24 data points to the models M_1 and M_2, we obtained

$$\boldsymbol{M_1}\textbf{: } E = 78.1 + 2.2 * S * EAF \quad (7)$$
$$\boldsymbol{M_2}\textbf{: } E = 43.9 + (2.8*Add+5.3*Mod+1.3*Del) * EAF \quad (8)$$

TABLE I. SUMMARY OF RESULTS OBTAINED FROM FITTING THE MODELS M1 AND M2

Metrics	M_1	M_2
R^2	0.50	0.75
β_0	78.1 (p =10^{-3})	43.9 (p =0.06)
β_1	2.2 (p =10^{-4})	2.8 (p =10^{-7})
β_2		5.3 (p =10^{-5})
β_3		1.3 (p =0.02)
MMRE	33%	20%
PRED(0.3)	58%	79%
PRED(0.25)	46%	71%

Table I shows the statistics obtained from the models. In both models, all estimates of the coefficients but β_0 on M_2 are statistically significant (p-value ≤ 0.05). The estimates of the intercept (β_0) in both models seem to indicate the average overhead for maintenance tasks.

The coefficient of determination (R^2) values suggest that 75% variability in the effort is predicted by the variables in M_2 while only 50% of that predicted by the variables in M_1. The MMRE, PRED(0.3) and PRED(0.25) values also show that M_2 outperforms M_1. M_2 produced estimates with a lower error (MMRE = 20%) than did M_1 (MMRE = 33%). Seventy nine percent of the estimates (19 out of 24) by M_2 have the MRE values of less than or equal to 30%. In other words, the model produces effort estimates that are within 30 percent of the actuals 79 percent of the time.

It is important to note that β_1, β_2, and β_3 are the estimates of coefficients of the *Add*, *Mod*, and *Del* variables, respectively. The β_1, β_2, and β_3 estimates in M_2 reflect the variances in the productivity of three maintenance types that we discussed above. These estimates show that *Add*, *Mod*, and *Del* variables have significantly different impacts on the effort estimate of M_2. One modified LOC affects as much as two added or four deleted LOC. That is, modifying one LOC is much more expensive than adding or deleting it. While M_1 treats *Add*, *Mod*, *Del* as having the same weight, the performance superiority of M_2 over M_1 can be explained by its use of separate LOC metrics, each having a different weight.

As shown in Table I, although *Del* has least impact on the effort as compared to *Add* and *Mod*, it is statistically correlated with the effort (p = 0.02). Thus, it is implausible to ignore the affect of LOC deleted on the effort.

VI. THREATS TO VALIDITY

A. Threats to Internal Validity

There are several threats to internal validity. The capabilities of the groups may differ significantly. We used a matched-within-subjects design [23] to assign participants to groups, which helped to reduce differences. In addition, we scored participants' answers on our pre- and post-experiment questionnaire about participants' C++ experience and understanding of the program. We performed *t-test* to test the differences in scores among the groups. The result indicated no statistically significant difference between any two groups (p > 0.23).

Another threat is the accuracy of time logs recorded by participants. We told participants to record start, end, and interruption time for each maintenance activity. This required participant to input their time consistently from one activity to another. In addition, we used the hardcopy timesheet instead of the softcopy one as we believe that it is more difficult to manipulate the time in the hardcopy and if manipulations were made, we could identify easily. The time data was found to be highly reliable.

A third threat concerns possible differences in complexity of the maintenance tasks. As the complexity is one of the key factors that significantly affects the productivity of maintenance tasks, the differences may cause the productivity to be incomparable between the groups. However, we designed the tasks that involve the same set of methods and classes, ensuring that the groups have the comparable productivity. The submitted source code was found to be consistent with our design.

B. Threats to External Validity

Differences in environment settings between the experiment and real software maintenance may limit the generalizability of the conclusions of this experiment.

First, professional maintainers may have more experience than our participants. As all of the participants except the senior were graduate students, and most of the participants including the senior had industry experience, we do not believe the difference in experience is 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.

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.

VII. CONCLUSIONS

This paper has described a controlled experiment to assess the productivity and effort distribution of three different maintenance types including enhancive, corrective, and reductive. Our study suggests that the productivity of corrective maintenance is significantly lower than that of the other types of maintenance and a large proportion of effort is devoted to program comprehension. These results are mainly consistent with the previous studies on productivity of maintenance types [12] and effort distribution [9][24]. Indeed, our results suggest that it is more expensive to modify than add or delete a LOC.

We also described effort models for explaining and estimating the individual programmer's time on the maintenance tasks. The results show that the model with three independent LOC metrics (added, deleted and modified) is more favorable than the other model that uses the total number of LOC. This model better handles the variability in the productivity of three different maintenance types.

Despite the validity concerns that we have discussed in Section VI, our study suggests several implications for maintenance effort estimation and staffing. Effort estimation models for maintenance work that measure the size in LOC should use the added, deleted, and modified LOC as three independent parameters other than the simple sum of the three. That is, each of these LOC metrics has a different level of influence on maintenance effort and thus it should be weighted accordingly. To further verify the significance of the separation, we are collecting and calibrating the COCOMO maintenance model using these size measures. Another implication is that reducing business rules of the program does not come at no cost. Rather, it requires as much effort as does implementing new business rules of comparable size. Finally, the differences in effort distribution among the maintenance types that our results have shown suggest that task assignment is important to utilize human resources efficiently. For example, assigning highly application-experienced programmers to fault fixing may save more effort from code comprehension than assigning them to enhancement tasks because of the significant difference in the distribution of effort between these maintenance types.

ACKNOWLEDGMENT

We would like to thank the students who participated in this experiment, we also thank Marilyn Sperka and Qi Li for reviewing the early versions of this paper.

REFERENCES

[1] A. Abran and H. Nguyenkim, "Analysis of Maintenance Work Categories Through Measurement," Proc. ConJ on Software Maintenance 1991, Sorrento, Italy, pp. 104-113.

[2] B. W. Boehm. "Understanding and Controlling Software Costs", IEEE Trans. on S.E., 1988.

[3] T.M. Pigoski, "Practical Software Maintenance: Best Practices for Managing Your Software Investment", John Wiley & Sons, Inc., New York, NY, 1996

[4] B.W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, "Software Cost Estimation with COCOMO II," Prentice Hall, 2000.

[5] M. Hariza, J.F. Voidrot, E. Minor, L. Pofelski, and S. Blazy, "Software Maintenance: An analysis of Industrial Needs and Constraints", Proceedings of the Int. Conf. on Softw. Maint. (ICSM), Orlando, Florida, 1992.

[6] B.W. Boehm, C. Abts, S. Chulani, "Software development cost estimation approaches: A survey," Annals of Software Engineering, 2000.

[7] M. Jorgensen , M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," IEEE Transactions on Software Engineering, v.33 n.1, p.33-53, January 2007

[8] A. De Lucia, E. Pompella, and S. Stefanucci (2005), "Assessing effort estimation models for corrective maintenance through empirical studies", Information and Software Technology 47, pp. 3–15

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

[10] M. Jørgensen, "Experience With the Accuracy of Software Maintenance Task Effort Prediction Models," IEEE Transactions on Software Engineering, v.21 n.8, p.674-681, August 1995

[11] N. Chapin, J.E. Hale, K.Md. Kham , J.F. Ramil, W. Tan, "Types of software evolution and software maintenance," Journal of Software Maintenance: Research and Practice, v.13 n.1, p.3-30, Jan. 2001

[12] T.L. Graves, A. Mockus, "Inferring Change Effort from Configuration Management Databases", Proceedings of the International Symposium on Software Metrics, IEEE, 1998, Pages 267-273

[13] V. Nguyen, S. Deeds-Rubin, T. Tan, B.W. Boehm, "A SLOC Counting Standard," COOCOMO II Int'l Forum, 2007. DOI = http://csse.usc.edu/csse/TECHRPTS/2007/usc-csse-2007-737/usc-csse-2007-737.pdf

[14] A.J. Ko, H.H. Aung, B.A. Myers. "Eliciting design requirements for maintenance-oriented ideas: a detailed study of corrective and perfective maintenance". Proceedings of the international conference on software engineering ICSE'2005. IEEE Computer Society; 2005. p. 126–35.

[15] D.C. Littman, J. Pinto, S. Letovsky, and E. Soloway, "Mental Models and Software Maintenance", Journal of Systems and Software 7, 341-355, 1987.

[16] B.W. Boehm, "Software Engineering Economics," Prentice Hall, 1981.

[17] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models", IEEE Transactions on Software Engineering 21 (8), 1995, pp.674–681.

[18] T. Chan, "Impact of Programming and Application-Specific Knowledge on Maintenance Effort: A Hazard Rate Model".

Proceedings of IEEE Int'l Conference on Software Maintenance, 2008, pp. 47-56.

[19] V.R. Basili, "Viewing Maintenance as Reuse-Oriented Software Development", IEEE Software, v.7 n.1, p.19-25, January 1990

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

[21] J.J. Dolado, "On the problem of the software cost function," Information and Software Technology 43 (2001), pp. 61 – 72

[22] D. Port, M. Korte, "Comparative studies of the model evaluation criterions MMRE and PRED in software cost estimation research", Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM), 2008, pp. 51-60

[23] L.B. Christensen, "Experimental Methodology", 8th Ed., Allyn & Bacon, 2000

[24] R.K. Fjelstad and W.T. Hamlen, "Application Program Maintenance Study: Report to Our Respondents", In Tutorial on Software Maintenance, G. Parikh and N. Zvegintzov, Eds., IEEE Computer Society Press, Los Angeles, CA, 1983. pp.11-27.