Information and Software Technology 53 (2011) 682–691

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A controlled experiment in assessing and estimating software maintenance tasks

Vu Nguyen [a,*], Barry Boehm [a], Phongphan Danphitsanuphan [b]

[a] Computer Science Department, University of Southern California, Los Angeles, USA
[b] Computer Science Department, King Mongkut's University of Technology North Bangkok, Bangkok, Thailand

ARTICLE INFO

Article history:
Available online 20 November 2010

Keywords:
Software maintenance
Software estimation
Maintenance experiment
COCOMO
Maintenance size

ABSTRACT

Context: Software maintenance is an important software engineering activity that has been reported to account for the majority of the software total cost. Thus, understanding the factors that influence the cost of software maintenance tasks helps maintainers to make informed decisions about their work.
Objective: This paper describes a controlled experiment of student programmers performing maintenance tasks on a C++ program. The objective of the study is to assess the maintenance size, effort, and effort distributions of three different maintenance types and to describe estimation models to predict the programmer's effort spent on maintenance tasks.
Method: Twenty-three graduate students and a senior majoring in computer science participated in the experiment. Each student was asked to perform maintenance tasks required for one of the three task groups. The impact of different LOC metrics on maintenance effort was also evaluated by fitting the data collected into four estimation models.
Results: The results indicate that corrective maintenance is much less productive than enhancive and reductive maintenance and program comprehension activities require as much as 50% of the total effort in corrective maintenance. Moreover, the best software effort model can estimate the time of 79% of the programmers with the error of or less than 30%.
Conclusion: Our study suggests that the LOC added, modified, and deleted metrics are good predictors for estimating the cost of software maintenance. Effort estimation models for maintenance work may use the LOC added, modified, deleted metrics as the independent parameters instead of the simple sum of the three. Another implication is that reducing business rules of the software requires a sizable proportion of the software maintenance effort. Finally, the differences in effort distribution among the maintenance types suggest that assigning maintenance tasks properly is important to effectively and efficiently utilize human resources.

1. Introduction

Software maintenance is crucial to ensuring useful lifetime of software systems. According to previous studies [1,4,29], 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 work.

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 [5,18].

There has been tremendous effort in software engineering community to study cost-driven factors and the amount of impact they have on maintenance effort [6,20]. A number of models have been proposed and applied in practice such as [2,5,12]. 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 [2,21], others such as [5] use only LOC that is added and modified. Obviously, the latter assumes that the deleted LOC is not significantly correlated with maintenance effort. This

* Corresponding author. Tel.: +1 323 481 1585; fax: +1 213 740 4927.
E-mail addresses: nguyenvu@usc.edu (V. Nguyen), boehm@usc.edu (B. Boehm), phongphand@kmutnb.ac.th (P. Danphitsanuphan).

0950-5849/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.11.003

inconsistency in using the size measure results in discrepancies in strategies proposed to improve software productivity and problems in comparing and converting estimates among estimation models.

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. [9]. We chose to study these maintenance types because they are the ones that change the business rules of the system by adding, modifying, and deleting the source code. They are typically the most common activities of software maintenance. The results of our study suggest that the corrective maintenance is less productive than enhancive and reductive maintenance. These results are largely consistent with the conclusion from previous studies [2,17]. The results further provide evidence about the effort distribution of maintenance tasks in which program comprehension requires as much as 50% of maintainer's total effort. In addition, our results on effort estimation models show that using three separate LOC added, modified, and deleted metrics as independent variables in the model will likely result in higher estimation accuracies.

The rest of the paper is organized as follows. Section 2 gives a discussion on the related work. Section 3 provides a method for calculating the equivalent LOC in maintenance programs. The experiment design and results are discussed in Sections 4 and 5. Section 6 describes models to estimate programmers' effort on maintenance tasks. Section 7 gives some discussions on the results. Section 8 discusses various threats to the validity of the research results, and the conclusions are given in Section 8.

2. Related work

Many studies have been published to address different size and effort related issues of software maintenance and propose approaches to estimating the cost of software maintenance work. To help better understand and access software maintenance work, Swanson [31] proposes a topology that classifies software maintenance into *adaptive*, *corrective*, and *perfective maintenance* types. This topology has become popular among researchers, and the IEEE has adapted these types in its *Standard for Software Maintenance* [19] along with an additional *preventive maintenance* type. In their proposed ontology of software maintenance, Kitchenham et al. [22] define two maintenance activity types, *corrections* and *enhancements*. The former type is equivalent to adaptive maintenance type while the latter can be generally equated to adaptive, perfective, and preventive maintenance types that are defined in Swanson's and IEEE's definitions. Chapin et al. [9] proposed a fine-grained twelve types of software maintenance and evolution. These types are classified into four clusters *support interface*, *documentation*, *software properties*, and *business rules*, respectively listed in the order of their impact on the software. The last cluster, which consists of *reductive*, *corrective*, and *enhancive* types, includes all activities that alter the business rules of the software. Chapin et al.'s classification does not have a clear analogy with the types defined by Swanson. As an exhaustive topology, however, it includes not only Swanson's and IEEE's maintenance types but also other maintenance-related activities such as training and consulting.

Empirical evidence on the distribution of effort among maintenance activities helps estimate maintenance effort more accurately through the use of appropriate parameters for each type of maintenance activity and helps better allocate maintenance resources. It is also useful to determine effort estimates for maintenance activities that are performed by different maintenance providers. Basili et al. [2] report an empirical study to characterize the effort distribution among maintenance activities and provide a model to estimate the effort of software releases. Among the findings, isolation activities were found to consume a higher proportion of effort in error correction than in enhancement changes, but a much smaller proportion of effort was spent on inspection, certification, and consulting in error correction. The other activities, which include analysis, design, and code/unit test, were found to take virtually the same proportions of effort in comparison between these two types of maintenance. Mattsson [25] describes a study on the data collected from four consecutive versions of a 6-year object-oriented application framework project. The study provides evolutional trends on the relative effort distribution of four technical phases (analysis, design, implementation, and test) across four versions of the project, showing that the proportion of implementation effort tends to decrease from the first version to the forth, while the proportion of analysis effort follows a reversed trend. Similarly, Yang et al. [32] present results from an empirical study on the effort distribution of a series of nine projects delivering respective nine versions a software product. All projects are maintenance type except the first project which delivers the first version of the series. The coding activity was found to account for the largest proportion of effort (42.8%) while the requirements and design activities consume only 10.2% and 14.5%, respectively. In addition to analyzing the correlation between maintenance size and productivity metrics and deriving effort estimation models for maintenance projects, De Lucia et al. [13] describe an empirical study on the effort distribution among five phases, namely inventory, analysis, design, implementation, and testing. The analyses were based on data obtained from a large Y2K project following the maintenance processes at a software organization. Their results show that the design phase is the most expensive, consuming about 38% of total effort, while the analysis and implementation phases account for small proportions, about 11% each. These results are somewhat contrary the results reported in Yang et al.'s [32]. A more recent study reported by the same authors (De Lucia et al.) presents estimation models and the distribution of effort from a different project in the same organization [14].

A number of studies have been reported to address the issues related to characterizing size metrics and building cost estimation models for software maintenance. In his COCOMO model for software cost estimation, Boehm presents an approach to estimating the annual effort required to maintain a software product. The approach uses a factor named *Annual Change Traffic* (ACT) to adjust the maintenance effort based on the effort estimated or actually spent for developing the software [7]. ACT specifies the estimated fraction of LOC which undergo change during a typical year. It includes source code addition and modification, but excludes deletion. If information is sufficient, the annual maintenance effort can be further adjusted by a maintenance effort adjustment factor computed as the product of predetermined effort multipliers. In a major extension, COCOMO II, the model introduces new formulas and additional parameters to compute the size of maintenance work and the size of reused and adapted modules [5]. The additional parameters take into account the effects such as the complexity of the legacy code and the familiarity of programmers with the system. In a more recent model extension to estimating maintenance cost, Nguyen proposes a set of formulas that unifies two COCOMO II's reuse and maintenance sizing methods [28]. The extension also takes into account the size of source code deletions and calibrates new rating scales of the cost drivers specific to software maintenance.

Basili et al. [2], together with characterizing the effort distribution of maintenance releases, describe a simple regression model