Cost Benefit Oriented Analysis for Designing Optimum Quality Assurance Practices on Software Development

Md. Baharul Islam
Department of Multimedia Technology and Creative Arts
Daffodil International University
Dhaka 1207, Bangladesh,
baharul@daffodilvarsity.edu.bd

A.H. M. Saiful Islam
Department of Computer Science and Engineering,
Daffodil International University,
Dhaka 1207, Bangladesh
saifcse@daffodilvarsity.edu.bd

Ziaur Rahman
Department of Information and Communication Technology
Mawlana Bhashani Science & Technology University
Tangail 1902, Bangladesh
zia@iut-dhaka.edu

***Abstract*—Quality is the essential part software development in the competitive market. Software requirement, design, coding and testing are playing an important role throughout the software development period. Saving at any of the developing stages will greatly reduce the total cost of software development. The key point of this paper is to detect and prevent of defects at earlier stages of software development with the cost control and make trade-off between the quality and the cost. The purpose of this paper is to provide a recommended process to develop software cost estimates for software managers and system engineers. The process is a simplified approach that should be followed by cost estimation professionals. Organization can reach their objectives by adjusting best balance between software quality and cost.**

Keywords- Cost analysis; Software development; Quality assurance; Cost optimization

I. INTRODUCTION

We visited some software outsourcing company in Bangladesh for getting the real scenario about the software quality assurance practices. These companies are offshore software development and information and communication technology (ICT) consulting firm which develops software product, provides application and web development, web solutions and performs IT consultancy in various fields in over the globe. These companies define itself by emphasizing central focus on providing best services to valued customers. They offer efficient solutions to valued customers by integrating solutions into their business strategy, practices and tools. Their main focus is to help customers add value to their businesses through the services provided by them. They believe in mutually beneficial long term partnership with their customers and they significantly invest their resources on learning and implementing new technologies in the most innovative manner to enhance performance, promote efficiency and finally, add tangible values to the businesses of their customers. The focal point of all services provided by these software companies is to customer satisfaction and quality assurance policy. They are believing and practicing their long term mutually beneficial relationship with customers by establishing close partnership at both technical as well as management level and by understanding the customers' business focus, values, practices, and processes. Quality assurance policy ensures that all deliverable products are provided on time, kept within scopes, delivered with quality as agreed upon by both customers and the outsourcing companies; and thus they are ensuring value addition to the business of our customers. A time-boxing model for iterative software development proposed [1]. Since they have the vision "Value Added Off-shore Services" is to add measurable business value for their customers in addition to integrating technology to off-shore software development. They should emphasis on improving research methodology to ensure software quality.

A. Scope

This paper describes a recommended set of software cost estimation steps that can be used for software projects, ranging from a completely new software development to reuse and modification of existing software.

It can be used by anyone who has to make software cost estimate including software managers, system and subsystem engineers, and cost estimators. The document also describes the historical data that needs to be collected and saved from each project to benefit future cost estimation efforts at the organization. This document covers all of the activities and support required to produce estimates from the software requirements analysis phase through completion of the system test phase of the software life-cycle.

B. Method

The prescribed method applies to the estimation of the costs associated with the software development portion of a project from software requirements analysis, design, coding, Integration and Test (I&T), through completion of system test. Activities are including software management, configuration management, and software quality assurance, as well as other costs, such as hardware procurement and travel costs that must also be included in an overall cost estimate. The estimation method is based upon the use of multiple estimates, data-driven estimates from historical experience, risk and uncertainty impacts on estimates.

C. Factors

Software development involves a number of inter-related factors which affect development effort and productivity. Many of these relationships are not well understood, accurate time estimation of software development and effort. The project planning process is split into a number of separate activities. Estimation involves answering the following questions:

- How much effort is required to complete each activity?
- How much calendar time is needed to complete each activity?
- What is the total cost of each activity?

D. Project Cost Estimation

Project cost estimation and project scheduling are normally carried out together. There were many analyses on software effort estimation techniques such as comparative [2-4], discriminate [5], framework [6], statistical [7], performance [8], and quality [9-10]. The cost of software development is primarily the cost of the effort involved. So the effort computation is used in both the cost and the schedule estimate. However we have to do some cost estimation before detailed schedules are drawn up. These initial estimates may be used to establish a budget for the project or to set a price for the software development of a customer. There are three parameters involved in computing the total cost of a software development project:

- Hardware, software and maintenance cost
- Travel and training cost
- Effort cost (payment of software engineers)

For most of the projects, the dominant cost is the effort cost [11-12]. Now-a-days computers that are powerful enough for software development are relatively cheap. Extensive travel cost may be needed when a project is developed at different sites. The travel cost is usually a small fraction of the effort cost. Furthermore using electronic communications systems such as e-mail, shared web sites and videoconferencing can significantly reduce the travel required. Electronic conferencing also means that traveling time is reduced that can be used more productively in software development. Organizations compute effort cost in terms of overhead cost where they take the total cost and divide this by the number of productive staff. Therefore the following cost is the part of the total effort cost:

- Cost of office space, heating and lighting
- Cost of support staff such as accountants, administrators, system managers, cleaners and technicians
- Cost of networking and communications

II. METHODOLOGY OF DATA GATHERING AND ANALYSIS

A. Characteristics of Software Quality

Software has both external and internal quality characteristics. External characteristics where user of the software product is aware including:

- Correctness- The degree where a system is free from faults in its specification, design, and implementation.
- Usability - The users can learn and use a system.
- Efficiency - Minimal use of system resources including memory and execution time.
- Reliability - The ability of a system to perform its required functions under stated conditions whenever required—having a long mean time between failures.

- Integrity - The degree where a system prevents unauthorized or improper access to its programs and data. The idea of integrity includes restricting unauthorized user accesses as well as ensuring that data is accessed properly.
- Adaptability - The extent where a system can be used, without modification, in applications or environments other than those for which it was specifically designed.
- Accuracy - The degree where a system, as built, is free from error, especially with respect to quantitative outputs. Accuracy differs from correctness; it is a determination of how well a system does the job.
- Robustness - The degree where a system can be continue perform the function in the presence of invalid inputs or stressful environmental conditions.

It is the only one characteristic that users care about whether the software is easy to use or not. They also care about whether the software works correctly or not where the code is readable or well structured. Programmers care about the internal characteristics of the software as well as the external ones, and it focuses on the internal quality characteristics. They are including:

- Maintainability - The ease with which we can modify a software system to change or add capabilities, improves performance, or correct defects.
- Flexibility - The extent to which we can modify a system for uses or environments other than those for which it were specifically designed.
- Reusability - The extent to which and the ease with which we can use parts of a system in other systems.
- Readability - The ease with which we can read and understand the source code of a system, especially at the detailed-statement level.
- Testability - The degree to which we can unit-test and system-test a system; the degree to which we can verify that the system meets its requirements.
- Understandability - The ease with which we can comprehend a system at both the system-organizational and detailed-statement levels.

The difference between internal and external characteristics isn't completely clear-cut because at some level internal characteristics affect external ones. Software that isn't internally understandable or maintainable impairs our ability to correct defects, which in turn affects the external characteristics of correctness and reliability. Software that isn't flexible can't be enhanced in response to user requests, which in turn affects the external characteristic of usability. The point is that some quality characteristics are emphasized to make life easier for the user and some are emphasized to make life easier for the programmer.

B. Finding a Defect

Debugging consists of finding the defect and fixing it. Finding the defect is usually 90 percent of the work. Debugging by thinking about the problem is much more effective and interesting than debugging with an eye.

C. The Scientific Method of Debugging

When we use the scientific method for debugging; we must go through the following steps:

- Gather data through repeatable experiments.
- Form a hypothesis that accounts for the relevant data.
- Design an experiment to prove or disprove the hypothesis.
- Prove or disprove the hypothesis.
- Repeat as needed.

This process has many parallels debugging. Here is an effective approach for finding a defect:

i. Stabilize the error.
ii. Locate the source of the error (the "fault").
    a. Gather the data that produces the defect.
    b. Analyze the data that has been gathered and form a hypothesis about the defect.
    c. Determine how to prove or disprove the hypothesis, either by testing the program or by examining the code.
    d. Prove or disprove the hypothesis using the procedure identified in 2(c).
iii. Fix the defect.
iv. Test the fix.

TABLE 1. STEPS OF SOFTWARE ESTIMATION

Action	Description	Responsibility	Output Summary
Step 1: Gather and Analyze Software Functional & Programmatic Requirements	Analyze and refine software requirements, software architecture, and programmatic constraints.	Software manager, system engineers, and cognizant engineers.	Identified constraints Methods used to refine requirements Resulting requirements Resulting architecture hierarchy
Step 2: Define the Work Elements and Procurements project.	Define software work elements and procurements for specific	Software manager, system engineers, and cognizant engineers.	Project-Specific product based software WBS Procurements Risk List
Step 3: Estimate Software Size	Estimate size of software in Logical Source Lines of Code (SLOC).	Software manager, Cognizant engineers.	Methods used for size estimation Lower level and total software size estimates in logical SLOC
Step 4: Estimate Software Effort Software manager, cognizant	Convert software size estimate in SLOC to software development effort. Use software development effort to derive effort for all work elements.	Engineers, and software estimators.	Methods used to estimate effort for all work elements Lower level and Total Software Development Effort in work-months (WM) Total Software Effort for all work elements of the project WBS in work-months Major assumptions used in effort estimates
Step 5: Schedule the effort	Determine length of time needed to complete the software effort. Establish time periods of work elements of the software project WBS and milestones.	Software manager, cognizant engineers, and software estimators.	Schedule for all work elements of project's software WBS Milestones and review dates Revised estimates and assumptions made
Step 6: Calculate the Cost	Estimate the total cost of the software project.	Software manager, cognizant engineers, and software estimators.	Methods used to estimate the cost Cost of procurements Itemization of cost elements in dollars across all work elements Total cost estimate in dollars
Step 7: Determine the Impact of Risks	Identify software project risks, estimate their impact, and revise estimates.	Software manager, cognizant engineers, and software estimators	Detailed Risk List Methods used in risk estimation Revised size, effort, and cost estimates
Step 8: Validate and Reconcile the Estimate Via Models and Analogy	Develop alternate effort, schedule, and cost estimates to validate original estimates and to improve accuracy.	Software manager, cognizant engineers, and software estimators.	Methods used to validate estimates Validated and revised size, effort, schedule, and cost estimates.
Step 9: Reconcile Estimates, Budget, and Schedule	Review above size, effort, schedule, and cost estimates and compare with project budget and schedule. Resolve inconsistencies.	Software manager, software engineers, software estimators, and sponsors.	Revised size, effort, schedule, risk and cost estimates Methods used to revise estimates Revised functionality Updated WBS Revised risk assessment
Step 10: Review and Approve the Estimates	Review and approve software size effort, schedule, and cost Estimates	The above personnel, software engineer with experience on similar project, line and project management.	Problems found with reconciled estimates Reviewed, revised, and approved size, effort, schedule, and cost estimates Work agreement(s), if necessary
Step 11: Track, Report, and Maintain the Estimates	Compare estimates with actual data. Track estimate accuracy. Report and maintain size, effort, schedule, and cost estimates at each major milestone.	Software manager, software engineers and software estimators	Evaluation of comparisons of actual and estimated data Updated software size, effort, schedule, risk and cost estimates Archived software data

D. Analysis of data

Table 2 shows the cost to correct defect in different phases of software development in whole life cycle. As special technical skills are needed, such as those of database administrators, quality assurance specialists, human factors specialists, and technical writers, it becomes more and more important to plan organization structures carefully. Indeed, among the hallmarks of the larger leading-edge corporations are measurement specialists and measurement organizations. One of the useful by-products of measurement is the ability to judge the relative effectiveness of organization structures such as hierarchical vs. matrix management for software projects and centralization vs. decentralization for the software function overall. Here too, measurement can lead to progress and the lack of measurement can lead to expensive mistakes.

TABLE 2. COST OF DEFECTS/PRICE OF QUALITY

Phase	Relative Cost to Correct defect
Definition	$1
High-Level Design	$2
Low-Level Design	$5
Code	$10
Unit Test	$15
Integration Test	$22
System Test	$50
Post-Delivery	$100+

III. OUTCOME AND RECOMMENDATION

In the table 1, we have certainly observed that prevention of defects and detection of early defects is the major requirement to improve software quality. If the error is detected at later stages the cost is also increasing proportionally in order to fixing the bugs. Even the quality decreases if the errors are detected at later stages because fixing a bug at later stages may add another bug and cause system malfunctioning. Based on the scenario we shall propose for improving better balance between quality and cost based on analysis are as follows:

A. Improve Project SQA Processes

The SQA activity for process improvement requires:

- Understanding project and SQA processes
- Determining where inefficiencies or defects occur (root causes of defects)
- Recommending changes to project processes to improve efficiency or reduce defects
- Recommending improvements to eliminate the root causes of defects
- Recommending training courses for the project team

The purpose of this activity is for SQA to review existing project and SQA processes, report on efficiencies, and area for improvement, and identify processes that need to be defined. To improve project SQA processes, SQA needs to review and audit both project processes and SQA processes. This will ensure that project and SQA processes are consistent and compatible with one another. Process improvement may result in changes to the policy, processes, and/or procedures.

B. Measurements for Defect Analysis

In some sense the goal of all methodologies and guidelines is to prevent defects. For example, a design methodology gives a set of guidelines that is used for a good design. In other words, the design methodology aims to prevent design defects by guiding designer along a path that produces good and correct designs. However we can learn from actual defect data by Defect Prevention (DP) with the goal of developing specific plans to prevent defects from future occurrence. The main goal of DP is the reduction in defect injection and consequent in rework effort. It is best if suitable measurements are made such that impact of DP can be quantitatively evaluated. A project employing DP should be able to see the impact of DP in the injection rate on the rework effort of project. Both of these proper metrics have to be collected. Furthermore, suitable data needs to be collected to facilitate the root cause analysis for DP. The measurements needed for evaluating the effectiveness of defects and effort. The defect data is easily available if projects follow the practice of defect logging. To facilitate defect analysis for each defect, a fixed set of categories should also be recorded. For understanding the impact of DP on rework, the effort on the project needs to be recorded with suitable granularity such that rework effort can be determined. Specifically, for each quality control activity, the rework effort should not be clubbed together with the activity effort but must be recorded separately. Effort logging generally requires that each member of the project record the effort spent on different tasks in effort monitoring system. Frequently, different codes are used for different categories of tasks. For most of the major tasks, the effort is divided into three separate categories – activity, review, and rework. With this type of categorization, rework effort for each phase can be determined. The measurement about defects and effort are sufficient to do defect analysis and prevention, as well as quantify the impact of DP. However without the effort data, the impact of DP on rework cannot be determined.

C. Cost Benefit Analysis

$Cost_c$ of Practicing Current Process

$Cost_{im}$ of Practicing improved Process

Cost increase $= Cost_{im} - Cost_c$

$= \$1000 - \$1500$

$= \$500$

Gross Benefit = $[CDF_c – CDF_{im} + MC_c – MC_{in} + CP_{im} - CP_c]$
= $2500- $500+$2500-$500+$1000-$2000
= $3000

Net Benefit = Gross Benefit - $Cost_{in}$ - $Cost_c$
= $3000- $500-$1500
=$1000

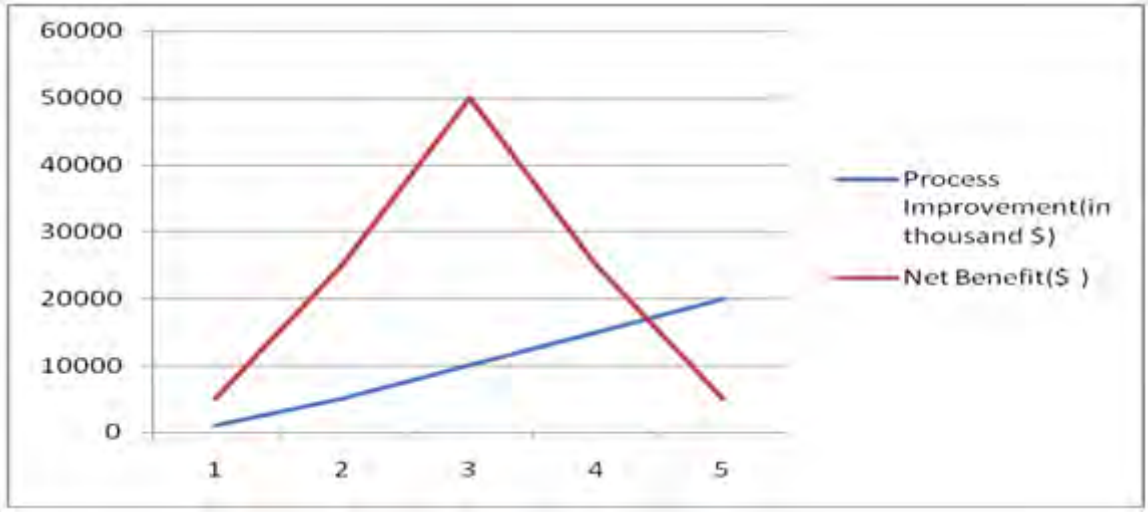

Figure 1. Net benefit vs. process improvement graph

From the figure 1 we noticed the net benefit and process improvement. Process improvement is gradually increased whereas net benefit increases exponential, and after reaching the pick point, it is dramatically fall down.

IV. CONCLUSION

The subjectivity and ambiguity of software measurement and metrics are troublesome today, but metrics problems are troublesome in every young science. The long-range prognosis for software measurement and metrics is guardedly favorable. Certainly function point metrics have continued to expand at a rapid rate. Software engineering will be an oxymoron unless the critical measurement problems are resolved. Without accurate metrics and effective measurement practices, programming will neither be able to advance beyond the level of a craft, nor take its place in the ranks of true professions.

REFERENCES

[1] P. Jalote, “Timeboxing: A process model for iterative software development”, Journal of Systems and Software, Vol. 70, pp. 117-127, 2004.
[2] P.K. Suri, P. Ranjan, Comparative Analysis of Software Effort Estimation Techniques, International Journal of Computer Applications, Vol. 48, No.21, pp. 12-19, June 2012
[3] S. Maheshwari1, P. Dinesh, C. Jain, A Comparative Analysis of Different types of Models in Software Development Life Cycle, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No. 5, pp. 285-290, May 2012
[4] Jovan Popović1 and Dragan Bojić1. 2012. A Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle. ComSIS Vol. 9, No. 1, January 2012
[5] Ahaiwe Josiah, Discriminant Analysis of the Effects of Software Cost Drivers on Software Project Schedule Estimates in Nigeria, Interdisciplinary Journal of Contemporary Research in Business, Vol. 4, No. 2, pp. 545-558, June 2012
[6] S. Grimstad, M. Jørgensen, A Framework for the Analysis of Software Cost Estimation Accuracy, Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE'06), September 21–22, 2006, Rio de Janeiro, Brazil.
[7] T.N.Sharma, A. Bhardwaj, G. R. Kherwa, Statistical Analysis of various models of Software Cost Estimation, International Journal of Engineering Research and Applications (IJERA), Vol. 2, No. 3, pp.683-685, May-Jun 2012
[8] N. Ramasubbu, R. K. Balan, Globally Distributed Software Development Project Performance: An Empirical Analysis, the Symposium on the Foundations of Software Engineering, ESEC-FSE’07, September 3–7, 2007, Cavtat near Dubrovnik, Croatia
[9] S. Kumaresh, R Baskaran, Defect Analysis and Prevention for Software Process Quality Improvement, International Journal of Computer Applications, Vol. 8, No.7, pp. 42-47, October 2010
[10] D. E. Harter and S. A. Slaughter, “Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis,” Management Science, vol. 49, pp. 784-800, 2003
[11] B. Boehm, C. Abts, S. Chulani, Software development cost estimation approaches – A survey, Annals of Software Engineering, Vol. 10, pp. 177–205, 2000.
[12] K. M. Flood, Reducing software development cost, schedule and risk using AGI software, Analytical Graphics Inc, pp. 1-14, 2009