Alphonce Omondi Ongere

Software Cost Estimation Review

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Alphonce Omondi Ongere Software cost-estimation review 53 pages 30th May, 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Marku Karhu, Head of Degree Programme

Software cost estimation is the process of predicting the effort, the time and the cost required to complete software project successfully. It involves size measurement of the software project to be produced, estimating and allocating the effort, drawing the project schedules, and finally, estimating overall cost of the project.

Accurate estimation of software project cost is an important factor for business and the welfare of software organization in general. If cost and effort estimation results into pessimistic estimate for a software project, suitable occasions can be missed due to imprecision. Optimistic predictions of software cost estimates can also lead to loss of some resources.

Over the past years the estimators have used parametric cost estimation models to establish software cost, however the challenges to accurate cost estimation keep evolving with the advancing technology. The need for researched solutions and continuous improvement of software cost estimation techniques and methods hold. Most cases of software cost estimation do not end up with accurate estimates as desired, but reliable estimates can be achieved.

Keywords	Software cost estimation process, Software cost estimate, Software organization staff

Contents

1 Introduction

Software cost estimation has become an issue of global economy [1]. Every software project manager, every software quality assurance specialist, system analyst and programmer should understand the basic concept of cost estimation [2]. Over the past decade a lot of changes has happened and continue to happen in the field of software cost estimation [3]. The increasing use of agile based development methods, object oriented methods, unified modeling language and use cases in software development have led to introduction of new metrics for estimation and measurements. These have added into the complexity of achieving accuracy of cost, schedule and effort prediction.

Currently, software is the driving force behind most day to day needs and service delivery such as education, business, entertainment, government operations, health facilities, military, and transport. Each of these sectors update, maintain or change to technologies that provide quality services to their clients. Most of these technologies are expensive, complex, and require accurate planning to be developed fairly fast.

The challenges to accurate prediction of cost, effort and schedule of software projects are equally growing. These challenges have to do with a variety of practical, measurement, and modeling factors. An approach to solve these issues calls for well defined, consistently applied and rigorously executed software estimation processes. The estimation process may be supported by techniques, models, or tools as will be established through this thesis. The question is, “what is the state of software cost estimation currently?”

According to Standish Group's Chaos summary report 2009, software projects have earned a reputation of a downward trend in project success due to delay, cost-overrun and project cancellation. It is easier to believe the Standish report, but a keen study of the critics analysis [4], only leaves an impression of some truth from the Standish group analysis. Other reports too add to the confirmation of the trend. [5; 6; 7.]

The goal of the project described in this thesis was to determine the current welfare of software-cost estimation processes and present a comprehensive and systematic review on software cost estimation techniques that can be of purpose and improve the criteria of achieving reliable software cost estimates. The objectives include establishing the current state of art for software cost estimation, analyzing the current estimation methods and procedures, determining immediate challenges and identifying the steps, procedures and practices that would improve cost estimation process.

This thesis includes an overview on the current cost estimation methods, a survey report carried out to establish the present state of art in software cost estimation, analysis on the challenges at hand, and a description of a comprehensive software cost estimation process. The thesis also identifies a set of software cost estimation steps that is applicable for software projects, ranging from completely new software development to maintenance and modification of existing ones. The steps and the methods can be used by anyone who wishes to make a software cost estimate, including software managers, entrepreneurs, system and subsystem engineers, and other cost estimators. The characteristics of a good estimate and factors that lead to reliable measurements are also described in this thesis.

2 Background and Related Work

This chapter presents an overview of the origins and evolution of software cost-estimation technology, and how software cost estimation fits within the broader category of software project management. In addition, it brings into focus the software development issues that affect software-cost estimation, some of which are investigated in current software organizations and discussed to illustrate the importance of this study.

2.1 History

The technology of software cost estimation started in the early 1960's when the estimation was manually performed and largely characterized by application of simple rules of thumb or local estimating algorithms developed through trial and error methods [8]. Increased computer usage prompted the need for developing large applications that would require justifiable cost estimates based on improved metric techniques alongside lines of code.

In 1973, Charles Turk at IBM built IBM's first automated estimation tool for systems software, and called it Interactive Productivity and Quality estimator (IPQ). This would later be renamed as Development Planning System in 1974 [9]. Later in same year, Dr. Randall Jensen at Hughes Aircraft developed a cost estimating methodology that grew later into the SEER software cost estimating tool [10]. In 1975 Allan Albrecht at IBM developed the original version of the function point metric based on five external attributes of software applications that are inputs, outputs, inquires, logical files, and interfaces [8; 11]. The function point would solve the complexity of variance across multiple programming languages, easing sizing and estimation of non-coding portions of software projects such as requirements, design, specification and manual creation.

Indeed, in the 1970's cost estimation methods were improved to cost estimation models, including predefined cost drivers that were then applied to obtain point estimates. A problem appeared in the selection of cost drivers from an increasing list of variables that were believed to influence software development efforts [11].

According to Conte [12], there was a need to focus on estimation methods that incorporated a combination of analytic equations, statistical data fitting and expert judgment. Conte's view was in response to the emphasis by early methods on project sizing, cost drivers or expert judgment. Conte's idea was shared by Barry Boehm and Larry Putman [13], who proposed the COCOMO and SLIM methods respectively. These methods considered the adjustment to nominal estimates by the experts and provided equations that incorporated system size as a principal effort driver. The predicted development effort would then be adjusted to accommodate the influence of 15 additional cost drivers.

1980’s were characterized by wide use of parametric methods [14] and application of standard function points measurements. This era created the nucleus of the current software cost estimation industry and marked the emergence of modern software cost estimation techniques. However, challenges such as inability to deal with exceptional conditions, proprietary algorithms and the ever changing relationships resulting into variation in productivity, led to the introduction and evaluation of non-parametric modeling techniques, such as artificial neural networks and analogy based estimation in the 1990’s. [14; 15.]

COCOMO II was published in 1995 to address the issue on non-sequential and rapid development process models, re-engineering, reuse driven approaches and object oriented approach. Most recently, researchers have turned their attention to a set of approaches that are soft computing based. These include fuzzy logic models, genetic algorithms [16] and others. The evolving challenges prompt the need for continuous systematic analysis and review of software cost estimation techniques and approaches, which is the main purpose of presentation in the subsequent chapters of this study.

2.2 Cost Estimation Methods

The methods discussed below have been selected and classified based on recency. Some of the methods describe the scenario of state of art in the immediate past and others the present state. The information on each method is publicly available, unambiguous and transparent. The older methods are classified as traditional cost estimation methods [17, 6] while the most recent approach to estimation methods is classified as modern cost estimation. This thesis investigates the relevance of the traditional software cost estimation methods as they have not only helped in the prediction of the project cost but in estimation and schedule drafting.
An illustration of three classes of software cost estimation methods is given below.

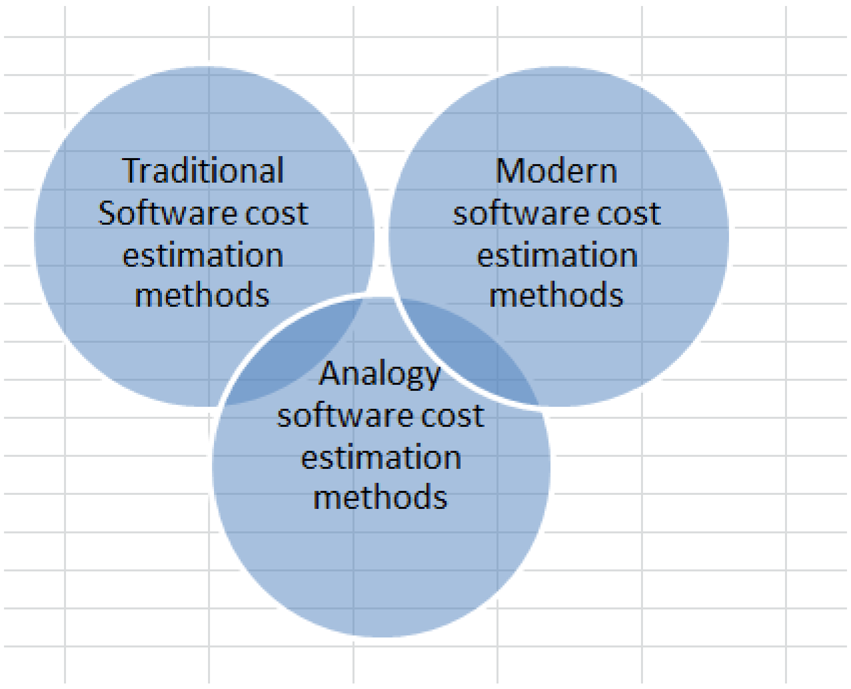
Traditional Software cost estimation methods
Modern software cost estimation methods
Analogy software cost estimation methods

Fig. 1.0 Classification of software cost estimation methods.

Based on the above diagram, analogy software cost estimation methods exist both in the traditional and modern estimation set-up. Some of the traditional and modern methods use the analogical approach to determine the size of a new project compared to an old one. Each of these methods is discussed in the next section.

2.2.1 Traditional Software Cost-estimating Methods

According to the recent software-cost estimation methods classification, traditional methodologies are grouped into three main estimation techniques [18]: empirical, heuristic and analytic software cost estimation techniques. Each of these techniques has various sub-models of estimation as briefly discussed below.

Empirical estimation techniques involve predicting the project parameters based on prior experience with similar development. There are different activities Involved in this method that have been formalized over the years. The popular empirical estimation techniques include expert judgment and Delphi cost estimation. [18, 45].

Expert judgment uses the experience and knowledge of a professional to come up with the cost of a planned project. In this technique, an expert provides the project size after analyzing a problem. [19; 20]. The expert estimates the cost of different components of the system to be developed, then combining every cost to arrive at the overall project estimate. This method seems to be easy to use and quite flexible. However, the method is subjective and prone to human error. It is also subject to omissions as the expert may overlook some factors inadvertently. Expert judgment is more refined when a group of experts are involved, as it helps minimize such factors as individual oversight, lack of familiarity with a particular aspect of a project, personal bias, and desire to win a contract through overly optimistic estimates. [20; 21.] Advantages of expert judgment include:

- It can be used where historical data are not available
- It is applicable in all acquisition phases of a project
- It can be blended with other cost-estimating methods within the same work-breakdown structures
- Experts may give a different perspective that might have been unknown or not considered

The disadvantages of expert judgment include:

- The objectivity is questionable
- It is not very accurate and cannot be used as a primary or basic cost estimation method
- It lacks supportive data or documentation

The Delphi cost estimation attempts to meet the shortcomings of expert judgment. It is carried out by a team of experts and a coordinator [22]. The coordinator provides each estimator with a copy of the software requirement specification document and a form for recording their estimates. The estimators complete their individual estimates separately without group discussions, and submit their results to the coordinator who prepares and redistributes the summary of estimates to all the estimators for further improvements. Based on the responses made by all estimators, the cost estimate is adjusted accordingly and iterated several times. The coordinator then prepares the final project cost estimate. [19; 20; 22.]

The advantages of the Delphi cost-estimating method include:

- Easy and inexpensive implementation
- Benefits from experience and knowledge of many experts
- Its usefulness for high level and detailed cost estimation
- Provide reliable estimates
- Delphi method tends to give a global view of projects to the team members

The disadvantages of the Delphi cost-estimating method include:

- There is a high chance of failing to reach a consensus
- Experts might give a biased estimate due to some circumstances
- It might prove difficult to work with different sets of teams
- It might lead to a false sense confidence with the estimate

Heuristic techniques are based on the assumption that the relationships among different project parameters can be modeled through a mathematical formula. When the basic parameters (independent) are identified, other parameters (dependent) can be substituted in the equation. Heuristic estimation models can be divided into two classes, namely single variable estimation model and multiple variable estimation models.

The single variable estimation models are represented by basic Constructive Cost Model – COCOMO [19]. Basic COCOMO was suggested by Dr. Barry Boehm [23]. According to him, any software development project can be classified into one of three categories, namely organic, semidetached, and embedded. The classification is done based on product characteristics as well as those of the development team and development environment.

A project is considered organic if it deals with development of a well understood application program, reasonably small in size and with experienced team members. A project is semi-detached if the development team consists of in-experienced staff or a team of staff unfamiliar with the project at hand. A project is embedded if the application is strongly coupled to hardware or the stringent regulations on the operational procedures exist [20]. The basic COCOMO takes the following equation form:

$$Effort = a1 \times (KLOC)^{b1} \text{PM} \qquad \text{eq1}$$

$$Tdev = 2.5 \times (Effort)^{c1} \text{ Month} \qquad \text{eq2}$$

Where:

- $KLOC$ is the estimated size of the software product expressed in kilo lines of code
- $a1$, $b1$, $c1$ are constants for each category of software products
- $Tdev$ is the estimated time to develop the software, expressed in months
- Effort is the total effort required to develop the software product, expressed in units person months (PM)

The values of the constants a1, b1 and c1 are shown in the table below.

Table 1.0 Parameters of basic COCOMO — reprinted from Merlo N [24, 4].

Basic COCOMO1	$a1$	$b1$	$c1$
Organic	2.4	1.05	0.38
Semi-detached	3	1.12	0.35
Embedded	3.6	1.2	0.32

Each mode has different values for constant depicting the weight of scale.

The following illustration shows an example of a single variable heuristic project cost estimate with a project size of 32 kilo lines of code.

KLOC = 32

$Effort$ = 2.4 × $(32)^{1.05}$ =91 person months

$Tdev$ = 2.5 × $(91)^{0.38}$ = 14 months

N = 91 ÷ 14 = 6.5 $people$

In the event that the average salary of a software engineer is € 3,500 then; the cost required to develop the project would be 14 × 3,500 = € 49,000

Multiple variable cost estimation models take the following form: Estimated resource = c1 x e1d1 x e2d2 + ….where e1, e2 are the basic (independent) characteristics of the software already estimated, and c1, c2, d1 and d2 are constants. An example of multiple variable models is Intermediate COCOMO model. Unlike the basic COCOMO model that assumes that effort and time development are functions of product size alone, the intermediate COCOMO model refines effort and time development using a set of fifteen cost drivers based on various attributes of software development. Each of the fifteen attributes are rated on a six point scale range of low to very high, and multiplied to attain total EAF (Effort Adjustment Factor). The effort and time equations get adjusted accordingly.

$Effort(nominal)$ = 2.5 × $(KLOC)^{b2}$ PM

$Tdev$ = 2.5 × $(Effort)^{c2}$ Months

Adjusted effort is corrected to:

$Effort = EAF$ × $a2$ × $(KLOC)^{b2}$ PM

The values of Intermediate COCOMO1 constants a2, b2 and c2 are shown in the following table.

Table 2.0 parameters of intermediate COCOMO —reprinted from Merlo N [24, 5].

Intermediate COCOMO	$a2$	$b2$	$c2$
Organic	3.2	1.05	0.38
Semi-detached	3	1.12	0.35
Embedded	3.6	1.2	0.32

The intermediate COCOMO model has three modes including organic, semidetached and embedded. The following example illustrates how it works. An organic type database system is designed for an office automation project, where the modules for implementation include data entry 0.6 KDSI, data updates 0.6 KDSI, query 0.8 KDSI, report generator 1.0 KDSI and system size 3.0 KDSI. The efforts are rated from low to high level as illustrated in the table below.

Table 3.0 Effort specification factors — adapted from Merlo N [24, 6].

Cost drivers	Level	Effort Adjustment Factor(EAF)
Complexity	High	1.11
Storage	high	1.02
Experience	low	1.13
Program capabilities	low	1.17

The project effort, the total time and the number of developers would be as follows.

$Effort = (1.11 * 1.02 * 1.13 * 1.17) * 3.2 * 3.0^{1.05}$

$Effort = 15.18\ Person\ months\ (PMs)$

$Tdev = 2.5 * (15.18)^{0.38}$

$Tdev = 7.02$ months

$N = 15.18 \div 7.02 = 2.2\ people$

Heuristic cost-estimating techniques are transparent and the cost drivers provide clarity on the various factors that affect project costs. Nevertheless, the technique's size calculation is difficult and the measurement unit vulnerable to miscalculation.

Analytical estimation techniques apply basic scientific assumptions about achieving the cost estimate. It begins with a few primitive program parameters to develop the expression for overall project length, potential minimum volume, actual volume, language level, and effort. The assumptions are best known for estimation of software maintenance efforts better than empirical and heuristic techniques. An example of analytical estimation technique is Halstead's software science. [25.]

Halstead's theory attempts to provide a formal definition and quantification of qualitative attributes such as program complexity, ease of understanding, level of abstraction based on low level parameters including operands, and operators appearing in the program. Although analytical methods are scientifically oriented, they can only be used in small projects. [25, 21.]

The illustration below summarizes the traditional software-cost estimation methods.

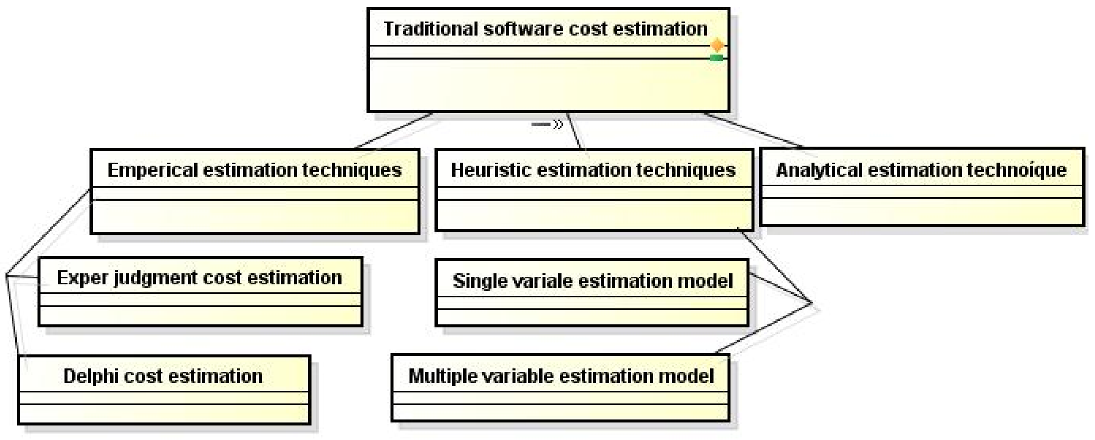
Traditional software cost estimation
Emperical estimation techniques
Heuristic estimation techniques
Analytical estimation technoique
Exper judgment cost estimation
Single variale estimation model
Delphi cost estimation
Multiple variable estimation model

Fig. 2.0 Traditional software cost estimation methods.

The figure illustrates three distinct traditional software cost estimation techniques, where empirical and heuristic techniques both have sub-estimation models.

2.2.2 Analogy Cost Estimation

Analogy software cost estimation predicts the amount of effort required to develop a system based on resemblance of the project's size, effort and productivity with the project under estimation, and it uses their data to derive cost estimate. It is a case based reasoning where similarities between the two projects are critical for determining the appropriate historical data to be used in deducing the estimate.

Initially the new project is characterized with attributes identical to the ones of completed projects registered in databases. The attributes must be quantitative such as implemented functionality in function points, and qualitative functionality such as programming language and application type used. Next, there is need to calculate quantity difference of the new project compared to the one at the database using Euclidean distance metric and based on the values of the selected attributes for these projects.

Analogy estimation is a common method that has been used by both traditional and modern cost estimation approaches. It is likely to stand the test of time as automated estimation models also use past historical data of completed projects to predict future ones. Analogy estimation can be applied in the early stage of development when requirements are fully unknown. [2,393]. However, there are some limitations such as the accuracy and consistency of the derived estimate depending on the quality of historical data, and whether the method is able to find analogies between the historical projects and the one being estimated [26].These limitations can be controlled when the method is calibrated to the local data and when it identifies projects that cannot be estimated with the analogy method. [26, 318].

2.2.3 Modern Software Cost Estimation Methods

Modern software cost estimation methods are classified on the basis of automation technology, manual techniques, software development methodology, project size and the complexity level. Capers Jones (2007) [2, 33] classifies the modern software cost estimation methods into classes and several sub-classes, namely manual software-estimating methods and automated software-estimating methods.

Manual software-estimating methods are sub-divided into three levels.

- Manual project-level estimation using rules of thumb
- Manual phase-level estimation using ratios and percentages
- Manual activity-level estimation using work-breakdown structures

The manual project-level cost estimation using rules of thumb constitute the oldest form of estimation which are still relevant in most software organizations to date. The project size is the key input into the estimation process function. Some examples include

- Raising the size of project application (measured in total function point) to the 0.4 power to predict the schedule of the project in calendar months from requirements until delivery
- A story that contains five story points can be coded in 30 hours of ideal time
- JAVA applications average 500 non-commentary code lines per staff month

[2, 34.]

The advantage of manual project-level estimation using rules of thumb is that they are easy to do. On the contrary, they cannot serve the purpose of signing contracts or formal budget for software projects.

Manual phase-level estimation begins with an overall project-level estimate, then assigning ratios and percentages to the phases such as gathering requirements, analysis and design, coding, testing, installation and training. The following example gives more details on manual phase-level estimation. An application of 100 function points in size, subjected to the phases mentioned above, would take assumptions of requirements comprising 10 percent of the effort, 20 percent of the analysis and design phase, 30 percent of the coding phase, 35 percent of testing, and 5 percent of the installation and training. [2, 34.]

The conversions of these percentages to actual effort translate into the results shown in the table below.

Table 4.0 Conversion of percentages into actual effort — reprinted from Capers Jones [2, 35].

Activities	**Effort**
Requirements	2 staff months
Analysis and design	4 staff months
Coding	6 staff months
Testing	7 staff months
Installation	1 staff months
TOTAL	20 staff months

The phase-level estimation methods using ratios and percentages are easy to do; however, they have some weaknesses which are discussed in the subsequent paragraph.

The percentages vary widely for every activity in reality, therefore it is not in order to use fixed percentages across all sizes of software projects; various software work span multiples phases or run through the entire length of the project. As an example, project management starts at the beginning of the requirements phase, and runs through the entire development cycle; some of the activities such as quality assurance cannot be identified as phases, hence risk omission.

Manual activity-level estimation using work-breakdown structures involves identifying key project tasks, and estimating the cost of each activity separately before summing up the total project estimate. It is so far the most reliable estimation method of all manual types, although it takes long to be done. The number of a normal range of software activities is 15 to 50 key deliverables. The activities, unlike phases do not assume a chronological sequence since multiple activities are found within any given phase. For example each software testing phase would have a number of testing activities such as new function, testing, regression testing, component testing, integration testing, stress testing and system testing. [2, 39.]

Automated methods are sub-divided into three levels [2, 36]

- Automated project-level estimates
- Automated phase-level estimates
- Automated activity/task-level estimates

The illustration below shows the modern cost estimation methods.

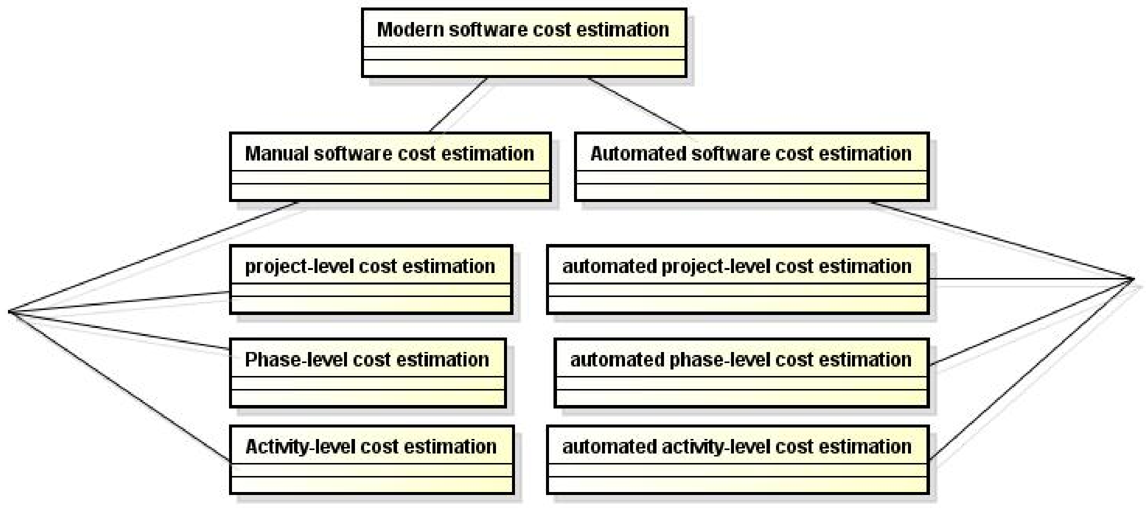
Modern software cost estimation
Manual software cost estimation
Automated software cost estimation
project-level cost estimation
Phase-level cost estimation
Activity-level cost estimation
automated project-level cost estimation
automated phase-level cost estimation
automated activity-level cost estimation

Fig. 3.0 Modern software cost estimation methods.

The automation methods are not different from the manual estimation methods, except that they are faster and easier to use. The automated estimation methods can be regrouped into macro-automated estimation methods and micro-automated estimation methods. Macro-automated estimation methods support two levels of granularity namely estimation of levels of complete projects and estimation of phase levels using built-in assumptions for ratios and percentages assigned to each phase. [2, 36.]

Since automated software-estimation tools are built on a knowledge base of so many software projects, the tools are better than manual estimation as they are able to adjust basic estimation equations in response to major factors that affect the project outcomes such as adjustments for levels of staff experience, adjustments for software development processes, adjustments for specific programming language used, adjustments for size of the software application and adjustments for work habits and overtime.

Macro-estimation tools fall short of accuracy as they are prone to omissions of some activities that constitute reliable cost estimate. The methods produce estimates that are not granular enough to support detailed cost estimation. Therefore according Capers Jones (2007), detailed work breakdown structure software cost estimation method can be achieved through micro-estimation methods. [2.]

The micro-estimating tools create detailed work-breakdown structure for the project and estimate each activity level separately. When every task level estimate is ready, the estimation tool sums up partial results for an overall cost estimate of staffing, effort, schedule and cost requirements. The advantages of using activity-based micro-estimation methods include the suitability for contracts and budgets following its data granularity, errors tend to be local and do not affect every activity of the project, new or unexpected activities can be added as need arises, activities not performed for specific projects can be backed out and they are suitable for agile based projects.

The following table gives a summary of work break down structure of software cost estimation methods, where estimation at a project level has no defined set of activities but rather ballpark estimation, while phase level estimation has six phases of project development and activity level estimation includes twenty-five sets of activities for a normal software project.

Table 5.0 Project, phase and activity- levels deliverable — reprinted from [2, 41].

Project level	Phase level	Activity level
Project	1. Requirements	1. Requirements
	2. Analysis	2. Prototyping
	3. Design	3. Architecture
	4. Coding	4. Planning
	5. Testing	5. Initial Design
	6. Installation	6. Detail design
		7. Design review
		8. Coding
		9. Reused code acquisition
		10. Package acquisition
		11. Code inspections
		12. Independent verification and validation
		13. Configuration control
		14. Integration
		15. User documentation
		16. Unit testing
		17. Function testing
		18. Integration testing
		19. System testing
		20. Field testing
		21. Acceptance testing
		22. Independent testing
		23. Quality Assurance
		24. Installation
		25. Management

Based on the table, it is arguable that the level of granularity depicts the precision for software estimate. At the activity level, the project is broken down into several tasks, while at the project level the project is estimated as a whole.

Despite the introduction of many cost estimation methodologies that an estimator may choose from, software cost-estimation accuracy remains a challenging task. The difficulty of finding a concise set of factors affecting estimation is attributed to lack of active research in key areas. Some these factors are described below while others are identified and discussed in the survey report (see chapter 3).

2.3 Cost Estimation Issues

Every cost estimating process follows a particular sequence of activities to arrive at an estimate. Nonetheless, there are issues that affect accurate cost estimation. Some issues that have been identified are discussed below. Their impact on current software-cost estimating processes are investigated and reported in the survey report (see chapter 3).

2.3.1 Requirement Issues

Software requirements are the starting point for every new project, and are a key contributor to enhancement of a project. Requirements are a specification of what should be implemented. They describe the behavior and attributes of a system and also lay the foundation for all subsequent project work. Both software sizing and software cost estimates are derived from the requirements themselves, so the precision with which requirements are defined affects the accuracy of the software size and cost estimate. [2; 12.]

Problems in requirements obviously lead to incorrect estimation. Many errors from requirement specification pass to other levels undetected leading to complications of fixing the errors. According to Kishore and Naik (2001) [27, 49] "a requirement defect is 100 to 500 times more expensive to fix once the software is in the field than to fix it at requirement level".

Although many analyses have tried to improve software requirements, the quantification of requirement size, schedules, effort and cost and also quantification of requirements errors and defect removal efficiency have been missed. [2, 368.]
The project described in this thesis investigates and reports what the software organization is doing to contain and handle the challenges of requirements in the current development environment.

2.3.2 Software Sizing Issues

Software cost estimation begins by prediction of the sizes of deliverables to be constructed. Software sizing is the process of determining how big an application to be developed will be. The software sizes rely on a number of factors. For instance, complex programs that perform many functions and require high reliability are typically bigger than simple projects. Size estimation requires a clear knowledge of the project scope, complexity and interactions. [28.] Size can be predicted in several ways including

- Size prediction using an estimating tool's built-in sizing algorithms
- Sizing by extrapolation based on the requirements' function point total
- Sizing by analogy from similar projects of known size
- Guessing the size using project manager's intuition
- Sizing using statistical methodologies
- Guessing the size using programmer's intuition

[2, 9.]

With several sizing approaches to choose from, the estimators must put into consideration factors such as the sizing technique having been rigorously defined and in a widely accepted format, the technique being consistently updated by an independent body, availability of data to support the continuity of counting by certified counters [28, 130]. Some challenges that are associated with it include the fact that it is performed in a variety of different contexts, with many choices of programming languages and structures used to specify the requirements and design. In addition, most projects have a combination of new, reused or modified components. Lastly there is still challenges with the continuous change of sizing deliverables that might differ with time, so that the original size is modified at a new time.

2.3.3 Software Metric Issues

Software metrics are an integral part of cost estimation in software engineering. They entail continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information. Based on all of the possible software entities and all the possible attributes of each of those entities, there are multitudes of possible software metrics.

Any measurement program should be based on a comprehensive measurement plan including the purpose of the metrics on product, process, resources and project goal; the task to be measured such as resource attributes, project features and processes characterized quantitatively; defined processes and sub-process where measurement is necessary; and lastly the manual or automated technique's for metrics capture. [29.] There are still challenges including difficulty in choosing the right metrics for a project and many more. However, up until recently there have not been enough studies that directly address the problems of metrics in object oriented software, function point derivatives and metrics conversion. Such are analyzed in the next chapter.

2.3.4 Software Complexity Issues

Complexity is the extent to which system design or implementation is difficult to understand and verify. Complexity in software cost estimation affects a number of independent and dependent variables that influence the cost-estimation of software projects. Project complexity influences the choice of development personnel leading to a small team for a project, hence controlling the schedule. It is always said that a small team yields higher productivity rate per head than a large team, but that is worth investigation in this study. From the application type, an appropriate value for system complexity can be determined. Complex software cost more, have more defects and are always challenging to update safely. The complexities affect a wide cross section of activities and results in a number of cases mentioned below.

- Lengthened development schedules
- Increased levels of bugs and defect rates
- Lower defect removal efficiency rates
- Decreased development productivity rates

- Increased maintenance staffing needs
- Increased need for more test cases

[2, 247; 30,139.]

The most common complexities identified based on Capers Jones (2007) especially while using the software sizing and automated estimation tools include [2].

1. Algorithmic complexity that concerns the length and structures of the algorithms for computable problems and affects development quality, development productivity, and maintenance productivity.

2. Code complexity that concerns subjective views of development and maintenance personnel about whether the code they are responsible for is complex. The opinions are used for calibration of formal complexity metrics such as essential and cyclomatic complexities.

3. Data complexity that deals with a number of attributes associated with entities. It is a key factor in dealing with data quality lacks metric parameter for evaluation and is only done through subjective ranges.

4. Essential complexity supported by variety of software tools and is often applicable as a warning indicator for potential software quality problems.

5. Function point complexity that refers to a set of adjustment factors needed to calculate function point total of a software project. It handles the variations in function point for example the U.S. Function points as defined by IFPUG (International Function Point User Group) has 14 complexity adjustment factors. The SPR (Standard Function Point) and feature point metrics use three complexity adjustment factors and The British Mark II function point use nineteen complexity factors.

6. Problem complexity which deals with the subjective opinions of real people and is considered important in the calibration of objective complexity measures.

The following chapter presents the survey information that was carried out to establish the current welfare of software cost estimation.

3 The Survey

The survey is divided into two main sections. The first section presents the means used to acquire information, the materials used and the range of information technology organizations that were involved. The second section presents the report on how the process was conducted and the lessons learned. Additionally, this section presents the identified challenges facing current software cost estimation and lastly describes the current practices in software industries. The survey objectives therefore include

- Determining the current state of the art in software cost estimation through software companies in Finland, the methods for predicting cost, effort and schedule of software systems
- Identifying common factors affecting the cost estimation on software projects from different software companies through their representatives
- Identifying steps for improving software cost-estimating practices based on current technologies
- Identifying research directions for estimating and controlling software costs.

3.1 Information Gathering

This phase of the survey was meant for collecting data from different organizations. The objective was to determine the current welfare of cost estimation based on observation of individuals (Information Technology students) and staff of software organizations. It aimed at identifying any common opinion about software cost and challenges across board. The expectation therefore, was of a range of cost-estimating factors due to each groups' varied views.

The primary approach began with discussions amongst friends and fellow students on what views they had about the accuracy of software cost-estimations. Further, an example was given of a student developing a game application and selling it to the companies at prices that include maintenance costs, and equally competitive in the development market.

Information was also collected from the social network for example Facebook. Most software organizations have Facebook network that are managed by their respective administrators.

The administrators were contacted through chat communication, and they responded by providing links to information of the related questions. The criteria for choosing who to ask about the subject were random and the experience lead to the conceptualization of coming up with a questionnaire for collecting data henceforth.

Following the varied forms of data gathering, a total of three questionnaires were used. The first was influenced by the questionings from friends and chats with the organization staff. The second and the third questionnaire were redrafted as a result of subsequent advice about the questions by the organization staff. They helped in re-framing the question to achieve the desired objectives.

3.2 Questionnaire

The purpose of the questionnaire was to help establish the state of art in software cost estimation, the significance of cost estimation on software development companies and the challenges facing accurate software cost estimation in the industry. The research findings will be of great assistance to young entrepreneurs and future software cost estimators. The questions used are objectively formulated but also meant to accommodate variations of responses.

The following are the first questionnaire questions that were sent to software organizations.

- What kind of software projects is the company involved in?
- When does the company carry out software cost estimation, and why?
- Can estimates be changed once decided?
- What is included in software cost?
- What method of software cost estimation does the company employ?
- Where is the project executed?
- Are the projects estimated, primary projects or part of an entire system?
- What are the challenges encountered in the project cost estimation?
- Does the estimation involve primary phase and final phase in cost estimation?
- In which phases of development are the estimates and re-estimates done?
- Does the company define cost estimate?
- What are the components or the elements of a cost estimate?
- What is the importance of cost estimate to the company?
- Do parametric models play any role in influencing the company’s estimation methods?
- Advice on critical areas to consider or researched for improvement on software cost estimation.

The questions above would determine a range of software projects under development and from these projects, information on cost estimation would be derived. Many companies are involved in different types of software projects such as web-based development projects, internal information systems, external outsourced projects, system software, embedded software projects, commercial software projects, and military software project. These projects have different estimation and processes.

The projects have different reasons for estimating the cost, time and schedule. The questionnaire also sorts information that identifies the constraints and limitations that affect software industries currently through a number of questions, for example 3, 6, and 8. The questionnaire was to identify the steps for improvement and research directions, as represented in the last question. In addition, it was meant to identify the steps that an organization follows to arrive at an estimate as depicted by the sequencing of the questions.

There were concerns about the nature of the questions and worries that the questions might be against the company information policy, and the fact that organizations would not give out their secrets about their operations lest such information fall in the hands of competitors. The other factor was lack of statement of assurance of anonymity. No organization wishes to be on public record without agreements and especially if they are not directly involved in the publishing of information. These reasons prompted the questionnaire changes and led to the redraft of the subsequent questionnaire. The second and the third questionnaire were quite similar with minor differences in some questions, but set to achieve same goal. Therefore, only one of the two questionnaires is represented below.

The purpose of the questionnaire was to help establish the current state of art in software cost estimation in Finland, the significance of cost estimation on software development companies and the challenges facing software cost estimators at present and future. The research findings will be of great assistance to young entrepreneurs and future software cost estimators. (Full anonymity is assured.)

1. Are there enough tools to make accurate cost estimates in software project development?

2. How do the original estimates and actual estimates of software projects compare in the end of a project?

3. Reasons to support your choices?

4. What would you suggest to be done to better the situation?

5. Are there any alarming risks on software cost estimation process, if it remains as it is currently?

6. What’s your view on the evolving sizing methods with regard to accuracy in cost estimation? Is it a positive or negative move? Any reasons.

7. For what purpose do cost estimates serve currently?

8. What are the causes of inaccurate cost estimations currently?

9. When does the company carry out software cost estimation, and any reasons for the timing?

10. Can estimates be changed once decided and are there any formalities?

11. If you were to recommend a particular software-estimation method, which ones would they be?

12. Do software companies insist on project data collection? Any importance?

13. Are there any forms of complexities affecting the outcome of software projects?

14. Which metric units are relevant currently?

15. Current trends that affect or influence the estimation of effort, schedule or cost of software projects, any merits and demerits?

16. What is your opinion on analogy estimating methods, parametric methodologies, expert judgment and rule of thumb methods respectively?

The questionnaire was designed on the basis of the objectives of the study, establishing the current state of art for software cost estimation, analyzing the current estimation methods and procedures, determining immediate challenges, identifying and discussing the criteria steps, procedures and practices that would improve the cost estimation process. The issues discussed in section 2.3 were also investigated.

The first question was to determine the relevance of the study, and if it was in order to conduct the survey for the improvement of software-cost estimation. Another goal was to prove that many software projects are still delayed, costs are overrun or projects get canceled even though that is out of public knowledge. The responses to the first question would also encourage proceeding with other questions as stated in the objectives. The second and third questions would help establish the current state of art, identify factors that have contributed to either the success or failure and also give way to continue with subsequent questions such as the fourth and fifth question.

The fifth question targeted views on the current software process and how it links with development design methods such as agile or sequential waterfall, and also to identify which one is mostly used in project development, and what the advantages and disadvantages of the methods are. Again the question was meant to identify any future improvements on the process as suggested by the interviewees.

Some issues raised in the second chapter of this study are reflected in question six. For example for cost estimate to be done, there must be measurement, but over the past, measurement has been evolving with introduction of new metric units, so this question targeted knowledge on new metrics units in the organizations and their significance on measurement of project size.

The importance of software cost estimation could have increased or changed from the purpose they have always served before, following the advancing of software technology. Question seven targeted the importance of software-cost estimation and new directions for research and control of software costs.

To identify ways of improving the cost-estimating practices and processes, it was necessary to identify first the current challenges, and find out what should be done or is being done to mitigate the challenges, hence the relevance of question eight.

The next question was meant to determine whether the cost-estimation approach by the companies is iterative or not. Similar responses from most organizations would reveal the preferred or most applicable software-cost estimation process.

The tenth question aimed to establish whether the cases of inaccuracy have completely changed despite the recommendations and improvements so far. These questions were designed to achieve overlapping objectives and in the assumption that they would attract divergent views and gunner more information from the sources.

The table below summarizes the design of the questions against the objectives and issues that were to be investigated;

Table 5.0 Questionnaire objectives and questions.

Objectives	**Questions**
To establish the current state of art of software-cost estimation	1, 2, 3, 6, 7, 9,16
To identify the common challenges affecting the accuracy of software-cost estimation	1, 2, 3, 5, 8, 12, 15
To identify steps and ways of improvement in software-cost estimating practices, and procedures	4, 5,9,10 ,15
To establish research directions	1, 2, 3, 4, 5, 11
Complexity issues	12, 13
Requirement issues	9, 10, 13
Size issues	6,13
Metric issues	13,14

Based on the table there was more emphasis on finding out the welfare of cost-estimation, the challenges and ways and means of improvement.

3.3 Interviews

The interview was the implementation of the designed questionnaire. It involved first making a phone call to the software organizations that were identified, explaining the mission of the study after which they were requested to participate. Next, the questionnaire was sent via e-mail and another phone call was made to confirm the reception and to enquire if they would agree to a meeting or whatever arrangement suitable.

Some staff from the organizations opted for a phone call interview due to their busy schedule but others booked a date for the meeting. The interview was done in a question and answer manner and short notes were taken that would later be reviewed and redrafted. Occasionally, the interviewed staff had to be re-contacted for extra clarifications. However, there were some challenges such as poor network connectivity that interfered with the flow of communication and resulted into interrupted collection of information.

The phone interview was mostly objective and not very elaborate due to the need of more coverage in a short period of time. The interviews had to be repeated for clarity and extra inquiries had to be made as the questionnaire contents changed too following the advice from the organization staff, also when more questions arose from some of information or advice they gave. Other challenges over the phone were voice mails and answer machines, bureaucracy from one department to another (most companies filter information that leaves from the organization), unanswered calls and delay of information gathering due to change of appointments by the personnel that were suddenly busy at the appointment time.

Meeting interviews were successful in timing and with more detailed information. They too were based on the questioning and answer approach as indicated in the questionnaire, but also the conversation that ensued and discussions on related issues. Although both the interviews provided substantial information, there was little reference materials available and so the reliability of information could only be based on the staff‘s experience and the knowledge about the topics.

However, not all questions received equal attention as the personnel claimed little or lack of knowledge at certain instances for example the question as to whether the estimated cost matches the final cost of the completed projects?. The answers would range from “that’s the work of management”, to “it does not matter” and sometimes to “I have never checked”. In addition, consultative discussion could not be done to facilitate the necessary information. Another limitation encountered was the lack of conversation recording during the interview. The need and usefulness of such information were realized later during redrafting and verification.

3.4 Other Sources

Social media is currently a popular medium of information exchange and many organizations have accounts too. The common social media are Twitter and Facebook. Some organizations were approached through chat communication offered by the social network. The organizations’ staff shared their ideas and further provided references to their online community information. They also suggested some useful links that contained relevant information which required membership registration first.

3.5 Organizations Involved

While carrying out this study, a total of fourteen software organizations from Finland were approached, most of them involved in different development activities. These organizations were selected from the 2013 survey report carried out by the Great Place to Work organization which ranks the best companies based on their research standards annually. [31]. The aim was to cover a wide range of software project development in order to get a comprehensive coverage that would bring out clear similarities on common issues, different challenges and varied solutions. Some of the aspects that the organization involved are listed below.

Mobile software development: these are organizations that develop applications for low power devices and sell them at the stores. Their interest in estimation involves the development costs of general applications, development costs of business applications, continuing costs after development of applications, mobile enterprise application platform costs and software costs of subscription services.

Embedded systems development: embedded systems deal with a wide range of life-cycle and business-supporting system applications. Their interest in estimation includes combination of cost pressure, long-life cycle, real time requirements, reliability requirement and design-culture dysfunction. [31.]

Collaboration solutions development: these organizations deal with corporate websites, intranet and extra-net solutions, and also document management solutions. Their interest in estimation includes high cost of implementation, reliability issues, and cost overruns due to optimism, advance degree of difficulty and cyclomatic complexities. [31.]

Commercial software development: these organizations are interested in estimating the time for completion for market purposes and competition. Some of the developments include e-commerce developments, e-services development and information management using open source. [31.]

Web development and mobile service: such organizations deal with system application, user interface developments, web services and also embedded services developments. Their interest in estimation includes cost of interactive features, administrative costs, costs of mobile operation, and costs of processing facilities. [31.]

Innovative development solutions: majorly involved in solution driven and professional services such as support services, construction management and low volume products to meet business needs. Their interest in estimation includes costs of site conceptualization, costs of project life cycle and costs of contracts and engineering. [31.]

Maintenance: the maintenance organizations involve in upgrading the large and existing systems. Their interest in cost-estimation is determining time and the contents of release applications. [31.]

Procurement: they are mainly involved in identifying the need for a system that requires some amount of software development and contract out to other organizations. Their interest in estimation is expected contract values for tenders. [31.]

4 Results

Out of the fourteen software organizations approached, only six participated in the survey with three meetings and other three phone interviews. The results presented in this thesis, therefore, are based on a sample of six different organizations.

In response to availability of tools to make accurate cost estimation, all the respondents unanimously said there were no such tools, but some approaches were fairly successful for example agile methods of cost-estimation. According to the views of the organizations' staff, on a general scale most software projects were performing quite well in terms of schedule adherence, costs and effort management with acceptable inaccuracies from the estimations.

Project cancellation was referred to as a rare case, following the assessment approaches that are put in place and set for projects before they are adopted by the organizations for development. It was also clear that accuracy of cost estimation has no perfect definition except for approximation, which varies depending on the many factors that influence the development of a project. The following illustration depicts an outlook of software project cost-estimating with four sections, namely successful, delay, overrun and canceled software projects.

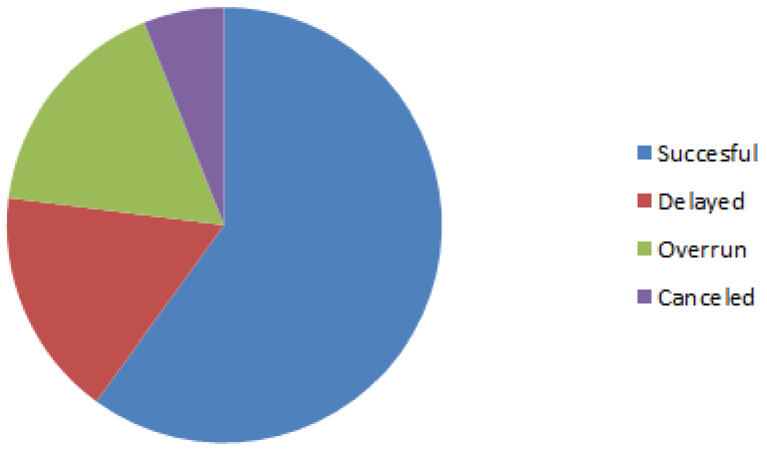

Fig 4.0 Software projects cost-estimation welfare.

From the illustration it can be seen that out of the six interviews the successful projects were those whose estimate costs compare closely to the actual costs and are within acceptable range, which is the case represented by the blue section of the graph.

However, delay and cost overruns tend to influence each other cost wise, so they are equally sectioned and represented with the red and green color respectively. It was noted that schedule could be kept as agreed to with the client but it also exerted pressure on effort and implementation. Thus an approach that is flexible, standardized and well defined depends on the choice of every organization and generalizations cannot be made.

Some of the factors that were identified and that have led to successful development of software projects in most organizations concerned the agile development approach which includes the following:

- Work breakdown structure with defined task-levels
- Priority of tasks
- Cooperation with all stakeholders
- Communication
- Discipline and well defined policies
- Honesty (especially with clients)
- Experience from similar projects

It was established that most cost-estimating methods have good plans to check cost overruns, delay and effort management except that between the start of a project and its release there is a long period of time and a lot of factors can interfere and eventually lead to unwanted outcome. During the interview, most respondents recommended the agile approach development as it aims at reducing cost and the development time, but also regulates the amount of work done.

They added that with agile development the project was developed iteratively, in very small increments and the solution is presented before a team for assessment. The key points noted under this approach of estimating were team collaboration, communication and management skills, determination to deliver a working solution or implementation in every sprint, client inclusions and also the open acceptance of requirement changes.

A work-breakdown structure is a decomposition of a project scope into the smallest level for implementation. It clearly shows the picture of the project in granular components, but the scope of the project must be determined first before any work-breakdown structure is established.

Every organization’s staff stressed the importance of the project scope which includes defined deliverable activities. The deliverables are further divided into tasks based on priority and the task-levels, to ensure appropriate time is spent on each task as projected in the project schedule.

It was established that part of the challenge with work breakdown structure despite its great contribution to accuracy in estimation would be the standard level of the tasks. Currently, there is no universal standard level for tasks, but every company has its own defined level. The following was the description of a good work breakdown structure.

- Should promote systematic planning process
- Include all possible project deliverables without omissions
- Simplify the project by dividing it into smaller and manageable elements
- Should be uniform and consistent

According to the interview reactions, other cost-estimating practices that led to schedule overrun resulted from lack of priority in the implementation. During the agile sprints, deliverables are sequenced in accordance to what is to be achieved and following the objective set per iteration. Priority of tasks enables the team work to be effective and also promotes the flow of development, “and that is healthy” said one staff.

Successful projects were achieved through the cooperation of every team member and decision making that was quick. A united team facilitated the implementation of agreements in the right time. Most challenges were approached with a combined effort. However, cooperation cannot happen without communication. Every organization’s staff reiterated that projects have failed to meet set goals, level of standards and even implementation due to lack of communication.

Communication is central to a good project management and a necessity for every project manager. Communication enhances clarity on the organization project plan, builds the relationships amongst the team members and promotes a friendly working environment. Communications promote good understanding between developers and the customers.

Management of a successful project goes further than just the project and the planning but it requires a lot of discipline. Lack of discipline and defined policies could compromise the scheduled plans and allow desperate actions which could lead to either cost overrun, delivery delay and effort mismanagement. An example given during a meeting was about a project manager who failed to insist on deadline or alternative ways to implement the plans every time disagreements came up, and it extended into the working time as it took several meetings to resolve the issues. The development time had been mismanaged and it ended in a fiasco. The view of nearly every respondent was that such situations were cases of “you don’t know what you are doing”.

4.1 Identified Challenges

The following challenges were identified:

- Requirements creep
- Expectation management
- Complex systems with different technologies
- Quality specification
- Developing new projects

Requirements issue was a common problem to all the organizations. According to the respondents, the customers tend to make changes or add more features different from original agreements but expect the terms of delivery, timing and cost to remain unchanged. These changes altered the scope of the project and the tasks which resulted in teams overrunning their original budget and schedule. The respondents felt that the problem was more customers centered therefore, organizations use ways such as holding joint application design so that the clients work side by side with the developers, freezing requirements for initial release at some point and moving additional requirements to subsequent releases. Lastly, including anticipated growth in the initial cost estimates.

The features for the final application may not be understood until a number of versions have been developed and used. The idea was supported by most respondents and they said it was effective. They added that Agile uses various forms of iterative development where pieces and final application are only developed and used after important features are understood.

Currently, the software organizations are experiencing cases where the customers say that the delivered product was substandard or is a different thing from what was requested. This is quite a difficult situation since it puts the organizations into huge loss following effort, resources and the time invested to develop a product. All respondents said that in such situations the clients’ money is usually returned. Therefore an agile cost-estimating approach would be the best solution.

Software projects are characterized on the basis of their complexity degree as some projects are simple and others complex. According to the survey respondents, the effort of a project is defined on the basis of complexity as it dictates the type of staffing to be assigned to a project. The current projects include new technological features that an organization may not have, leading to outsourcing of projects or collaboration with other companies that can handle them. A very complex project draws higher costs and might take a long time to develop if the staff is made up of few persons with knowledge and experience about the new technologies or trainees.

Quality specification is another challenge that affects cost estimation. There are standardized quality levels for evaluating the developed product and also the internal company-based standards. According to the survey, the number of development iterations that a product is subjected to improves its quality but the decision on the repetition depends on the management. If a project is subjected to too much iteration than planned, the schedule and the cost are affected and eventually the estimated cost turns out different from the actual cost.

Size and metric issues were also discussed during the survey interviews. It was established that most software projects today determine the size of a project in work amount basis. The amount of work is derived from the number of tasks and activities that must be done to implement functionality and it is measured in work-hours. The size is normally derived from requirements specification and used to determine the effort and the cost of the project.

In response to how software organizations dealt with early estimation with inadequate requirements to determine the total project size, the staff responded that agile approach was the best solution as it promotes incremental development where the size of a few deliverables could be determined using a few requirement information. Other respondents suggested the following:

- Pattern matching or use of historical data from similar projects. The method is limited to availability of data on schedules, costs and quality. Also the programming language may not be the same with that of new replacement.
- Using mathematical or statistical methods to derive the overall size of a project based on partial requirements. The method is limited to the knowledge of at least one factor and also availability of historical data from a similar project.
- Using rules of thumb to add contingency amounts to initial estimates to fund for future requirements. This method is limited to customers' approval on the contingency fund, otherwise psychologically unsettling.

Other organizations prefer user stories for measurement of the project size and draw the schedule and cost from it. Also function points were mentioned but less frequently, which is an indication of preference to the emerging and new sizing methods. The following are some of the sizing metrics including their advantages and disadvantages.

1. Source Lines of Code

Source lines of code metrics consider the volume of code required to develop the software project. They include executable instruction and data declarations, but exclude comments and blanks. They support cost-estimation by analogy, engineering expertise or automated code counters. The sizing is appropriate for projects preceded by similar ones, for example the same language or type of application. It is necessary to clearly define what is to be included during the development of code counts.

Source lines of code have some advantages for example, they can be used to estimate real time systems easily, manually or by automated code counter and large databases of historical program sizes are available, and they are easy to use. However, source lines of code lack a standard definition of what should be counted as lines of code, for example physical line or logical statements. Different lines of code count for the same function, depending on language and the programmer's style. The metrics emphasizes coding effort which is small compared to development effort and cannot be used for early estimation.

2. Function Points

Function point metric considers how many functions a program includes, types of input to the application, for example user inputs (add, change, delete), outputs (reports), data files to be updated by the applications, and inquiries (searches or retrievals) [2, 100]. Each function is weighed for complexity and count is adjusted for the effect of 14 characteristics such as data communications, transaction rate, installation ease, and whether there are multiple sites. Accurate counting requires in-depth knowledge of standards, experience, and preferably, function point certification. Function point analysis is linked directly to system requirements and functionality, so size analysis is measured in terms that a user can understand.

The size estimates can be based on quantifiable analysis through the project life cycle as requirements change. Function points are particularly useful in many development environments that might use unified modeling language, commercial off-the-shelf components, or object-oriented approaches to software development and implementation.

The advantages of function points include the application to most data types and that it can be used throughout development during interviews, requirements and design documents, data dictionaries and models, end user guides and also screen captures. Function point is independent of language or technologies and counts are available early in development from requirements and design specifications. The metric is standardized and often reviewed by the International Function Point Users Group. It is also used to determine the requirements creep. However, function points are not able to capture technical and design constraints and are subjective in counting. The use of function point also fails to derive requirements from a top level of specification [2.]

3. Object Point Analysis

Object point metrics use integrated computer-aided software engineering CASE tools to count the number of screens, reports, and third-generation modules for basic sizing.

CASE tools replace the manual writing of software code by using graphical user interface generators, libraries of reusable components, and other design tools. They emphasize actors involved in the solution and the actions they must take.

The advantages include support for inheritance as similar behaviors can be grouped into classes; support for re-use, and automation of manual activities. However, counting occurs only at the end of design and is not popular. Therefore, validated productivity metrics are not available.

4. Use Case Points

Use-case points metrics define interactions between external users and the system to achieve a goal, for example, a capture of a fingerprint or facial biometric to enroll applicants. A use case model describes a system's functional requirements. The model consists of all users and use case-tasks performed by the end user of a system that has a useful outcome, and identifies reuse by use case inclusions and extensions. Sizing count is arrived at by categorizing use cases as small, medium, or large and applying an average use case points per category. The addition of a complexity factor to the sizing count based on number and types of users and transactions improves the count accuracy.

The advantages include suitability for interactive end user applications and devices users interact with, increasingly applied to real-time systems that can be mapped to test cases and business scenarios, use case points are also intuitive to stake holders and team's plans and output. The shortcomings include being very likely to give inaccurate final estimates if system engineering process is immature and historical data lack counting standards. Using case points requires experienced object-oriented design team and estimates cannot happen unless defined use-case document is available.

5. Cosmic Function Points

Cosmic function point measures the size of software based on functional user requirements. The metric sizes the software project-independent of the technology to be used to implement it. Cosmic function points focuses on practice and procedures that software must follow to meet user needs. They are based on four different data movements namely entry, exit, read, and write. Each one constitutes a COSMIC function point. The method can be used to determine the software size of various applications including business, real-time such as telecommunications and process control, embedded software, for example cellular phones, electronics, and infrastructure software such as operating systems.

The advantages include simplified sizing since all data parts have the same value; comic sizing does not depend on data attributes. The counting standards are available and they apply to real time embedded systems. However, they are still new metrics and they cannot support bench marking. Also, the detailed information about data part takes long to be collected, which is less accurate for counting algorithmic software.

6. Story Points

A story point is an integer number that represents an aggregation of a number of aspects, each of which contributes to the potential of a story. A story point is based on the knowledge of understanding a problem, the level of complexity relative to ease of implementation, duration of implementation, uncertainty consideration and its potentials [26.] The advantages include its applicability in agile development, being easier to use without anxiety. A story point involves everyone in the project and therefore it is visible, it provides a logic way of structuring requirements and also based on the user's perspective. The disadvantages include taking too long on large projects following detailed discussions on its abstraction and the difficulty to determine quality specifications due to less experience in its application.

Some of the challenging areas identified during the study and recommended for further research include metric conversion, expectation management, quality specification, automating web development testing for quality and test-cost reduction, and lastly research on new developments and how to fit them successfully in a system.

4.2 Current Practices

Findings on trending habits associated with cost-estimating identified during the interview are described below. Outsourcing is one of the practices that most companies currently undertake to reduce the cost of project production. Reasons for outsourcing include cost savings, as they reduce work load on employees, time saving since the project is attended to round the clock, lack of in-house experience, flexibility and risk mitigation.

Most organizations that develop large system projects, achieve their target cost and schedule through outsourced services. They also emphasize the quality especially when the organization for which the job is outsourced is not an affiliate company. Some of the considerations made during outsourcing decision making include task importance. For example, tasks that are of core competence and critical success factors cannot be outsourced while tasks that are not core competence can be outsourced.

According to the survey findings, most respondents emphasized that it is best practice to have one manager for outsourced activities as it would create harmony and help achieve the set project objectives. For cautious purposes, it was pointed out that outsourcing could also be of some risk especially if the underlined policies and agreements, particularly on prioritizing are neglected. It can easily result into project delays or compromise the quality agreements.

Tendering is another cost-estimation related trend identified from the survey interviews. Tendering involves a procurement process based on the lowest price, and mostly practiced by the large software organizations that have many projects but lack time and in-house experience to accomplish the project. According to the interview respondents, this method risks quality of production, expectation issues due to misunderstanding, and project delay. Most delivery disagreements arise when the customer makes payments and receives the wrong product or the functionality of which might be unsatisfactory.

Lastly, the wide use of open-source developments by commercial organizations such as web developers enable small business enterprises to thrive at reduced production cost as the applications are free for use.

They also enable such small organizations that cannot afford highly skilled developers with readily developed application that can be modified and redistributed and in the process reduce their schedule time for delivery.

The method is most suitable for young entrepreneurs and school leavers who seek self-employment. However, open-source could have negative impacts on cost-estimation, for example adapting to ready applications and trainings might lead to a long time spent for preparation before actual development begins and this delay would give different costs compared to the original estimates. The business services relying on such a platform risk failing, if the company goes out of order.

4.2.1 Findings on Cost-estimating Methods

The following illustration presents an analysis on the software-cost estimation methodologies. Based on the methodologies discussed at the beginning of this study, the relevance and the use were to be determined and through this illustration, so that the most preferable and frequently used methods could be identified.

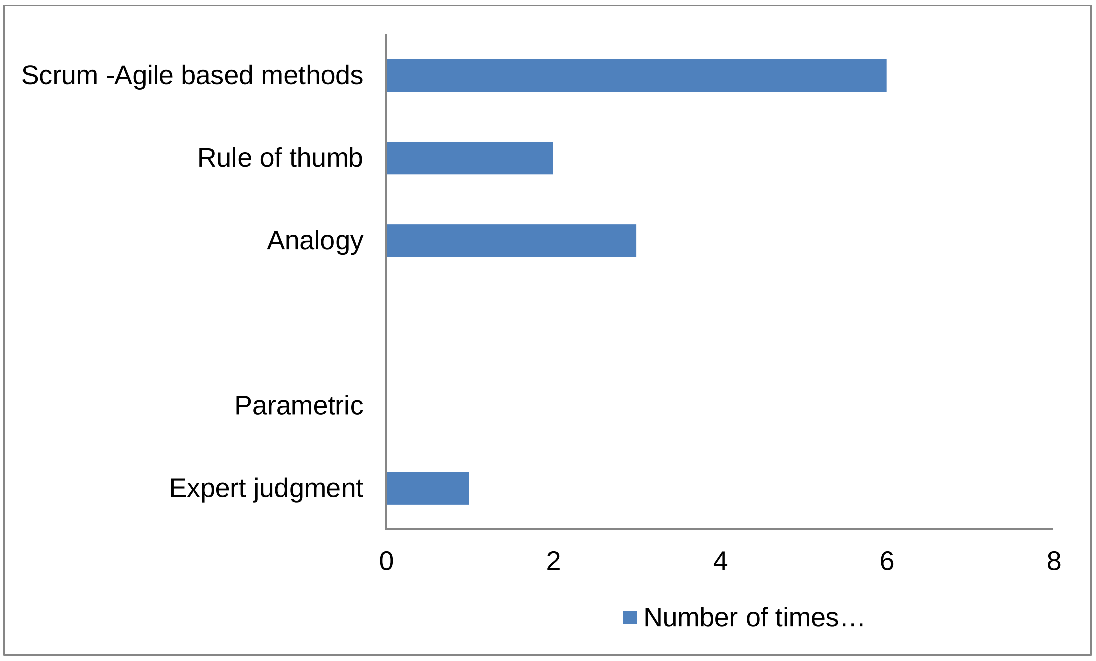

Fig 5.0 Cost estimation methods recommendations.

The illustration clearly shows that traditional cost estimation methods are rarely used and agile based methods are more preferred.

Although expert opinion is still popular, according to the personnel interviewed, the experts or experienced estimators are involved in the estimations at supervisory and consultative level rather than them doing the exercise by themselves. The agile method was suggested, and it was possible to deduce that most organizations use or prefer the agile estimation method. Some individuals argued that the methods and tools could be in place but accurate cost estimation might still be influenced by other unforeseen factors. Therefore they recommended the use of estimation process that promote activities and task definition with clearly defined scope of deliverable to determine software costs.

4.2.2 Findings on the Importance of Software Cost-estimation

The question as to whether cost-estimation is important was resoundingly approved by every respondent who took part in the survey. Some of the reasons cited were, that software cost-estimation is the basis for project bidding, budgeting and planning. Every organization does a rough cost estimate for any project to identify if it matches the available resources including technology skills before undertaking to develop it. Cost-estimating is essential for project management as it provides the framework for task allocation and other resources distribution.

The outsourcing organizations value cost-estimation to enable them to prioritize projects with respect to overall business plan. Also it is useful in assessing the impacts of changes and support project re-planning. Finally cost estimation helps in controlling the resource allocation.

The following are some of the opinion made by the respondents about the impact of automated testing for web development on cost estimation. The advantages and disadvantages are listed below.

Advantages of automated testing tools

- They are more efficient and provide repeatable environment.
- They can test graphical user interfaces, network communication, memory leakage, and can be calibrated to adjust to new test features.
- The tool is beneficial in product quality and minimizes the project schedule and effort through early test activity and test plan.

- They result in a reduced effort as the test engineers are only involved during business analysis and requirement activities and also analysis and design reviews.

Disadvantages of automated testing tools

- They are very expensive compared to manual testing tools, especially the initial investments.
- They cannot test everything; some areas need to be done by the testers, at least verifications.

5 Discussion and Recommendations

Both manual and automated cost-estimating methods should be able to provide cost estimate information at any level of project development. Although each of the methods have differences and involve varying activities during the development process, there is always a starting point and subsequent steps that reflect a framework for both manual and automated cost-estimation process. According to Caper Jones [2, 14] there are steps that would render reliable if not accurate cost estimates that satisfy the customer’s expectation and provide realistic goals to the developer. These steps are identified as software cost estimation sequence.

5.1 Estimation Sequence

The software cost estimation sequence consists of twelve steps that begin with requirements set to the delivery of the projects. They include

- Analyzing the requirements to create the project function point totals as basic size data to be used for formal estimation. This is recommended to be done by certified function point professionals [2, 8.] The data size can be expressed in terms of source code however; this would not reflect all project deliverables but only the code. Requirements analysis involves frequent communication with system users to determine specific feature expectations, avoiding feature creeps [2, 8.]

- Since feature creeps are inevitable in most occasions despite the measures taken during analysis, there is need to calculate the average growth rate for requirements which is planned for and included in the estimate. This is necessary for agile methods and iterative development projects [2, 9.]

- Deriving the size of project-key deliverables using the tools or any other suitable method. This can be done through sizing by extrapolation from function point totals (determined in first the step), sizing by analogy with similar projects of known size and sizing using expert judgment such as programmer or project managers [2, 9.]

- Once the sizes of key deliverable are known, selecting a set of activities to be performed is possible. Accurate software cost estimation is impossible without knowledge of the activities that are to be performed [2, 10]. The activities include requirements review, internal and external design inspections, coding and code inspections, user document creation, meeting sessions, change control integration, quality assurance, unit regression and system testing, and project management. The list of activities creates an impression of effort requirement and how the selected activities will be implemented [2, 10.]

- Every activity has a defect potential. The most expensive and time consuming work in the software development process is finding and fixing the bugs. Defect estimating at the activity level limits a series of reviews, inspections, and multi-stage tests costs. It is important to estimate both defect potential and defect removal efficiency. This can be done through predictive abilities for requirements defects, design defects, coding defects, user documentation defects, and bad fixes defects - injected while repairing previous defects [2, 10.]

- Estimating staff requirement that matches the project activities identified and is competent based on the average amount of work allocation. It is also necessary to identify the category of a worker and the numbers of workers for the overall project [2, 11.]

- Productivity rate depends on the abilities of the staff selected; therefore adjustments need to be made based on the level of experience of the skill factors of the development team [2, 11.]

- Estimation of effort and schedule would then be based on the size of the project, the activities identified, and the number of staff-team and their level of experience. Also the number of increments or sprints that will be done, the sizes of deliverables and the overlap between activities with mutual dependencies is considered for schedule estimation [2, 12.]

- Estimating the development cost based on effort, schedule, number of workers and their average salaries. If the project runs for several years, then inflation rates must be included. In the event that a project is developed internationally then currency exchange rates are taken into account. Other cost factors considered include license fees for acquired software, capitol costs for any new equipment, moving and living costs for new staff, travel cost for international projects at different locations of development, legal fees for copyright, patents, marketing and advertising costs [2, 13.]

- Estimating software maintenance and enhancement cost based on the probable number of users of the application, probable number of bugs or defects in the product at the time of release. This step requires the knowledge of good historical data on the rate of change of similar projects. For example, new software can add ten percent or more in the total volume of new features with each release for several releases in a row but does slow down for a period of two to three years before another major release [2, 14.]

- Presenting the cost estimate to a client for approval. The customer may agree or disagree. The estimator can only convince the client through a list of activities and tasks for the project or historical data from some similar project [2, 14.]

5.2 Software-cost Estimation Improvement

Effective monitoring and control of software development process is necessary for the success of a project. The project must be measured according to the appropriate level of granularity. The managers need to consider every task detail and level of the project to determine a clear scope. These tasks should have defined output and an objective method of determining their completion. The managers must also set the quality control at desirable levels. [2, 625; 32.]

Analysis of every report on problems encountered during development and tracing them from the work-breakdown structure and at the activity level should be encouraged. It enables quick inspections and means of determining the exact location of a project-problem efficiently. [2, 348.]

Software cost estimation should have a formalized process that defines when and how cost-estimation is performed. The process includes when cost re-estimation is done, and a clearly defined process of performing it. [2,622.]

Software organizations should encourage software reusability in the development process. A successful reusability program would depend on mastery of software quality and the technologies such as formal inspections. Also they can minimize requirements creep the agile way, by including the customer in the requirement specification process and allowing modifications during development process [29.]

Organizations should keep and maintain every project record database for future reference. The database should include the project metric units describing the features of the project, and the process metrics which describes the project development. [2, 422; 29.]

6 Conclusion

Following the objectives that were set during the onset of this review, it was necessary to conduct a survey to get reliable and precise information on software cost estimation from the software organizations. Though the exercise was challenging enough, the outcome led to the verification of the hypothesis and the learning of new information. A recap on some of the findings is given in the next paragraphs.

The search to establish the current state of art led to the conclusion that most software projects particularly in Finland are delivered on time, within acceptable cost limits and on adjustable effort levels. The state was majorly attributed to the current agile development approach that most organizations apply. This approach was learnt to provide multiple benefits to both the clients and the developers. For example, it not only focuses on customer satisfaction, cost reduction and shortening the production period, but it also offers a light weight framework for helping development teams realize the set target. Agile promotes iterative planning and feedback loops to enable teams to align the developments with necessary customer requirements.

It was discovered that currently software project sizing is relative to project work amount, as the size of a project is defined in terms of how much work hours it requires. The choice of project staff is based on capability, competency and professionalism depending on the category of projects. Some projects are crucial as the staffing might demand capability restrictions. Although most of the factors discussed so far are internal factors to the current state of art, other factors learnt include outsourcing and tendering which are aimed at production cost reduction, and quality achievements. In addition, some organizations and individual entrepreneurs exploit the open source opportunity legally to develop their products at free cost.

Despite all the success discoveries, there were some challenges too. Expectation management is an issue that continues to cause misunderstandings between clients and developers. However, many organizations compete to woo customers by promising quality products at low prices. Another challenge noted was quality specification that also encourages the market competition. It is close to impossible to define or specify quality standards in volatile technology environment. Many projects have been abandoned following their irrelevance with changing technologies and the growing complexities.

Lastly, the software cost estimation process stands insignificant with the increased application of the agile development method, but again estimates are necessary for any allocation of resource and time. Therefore, software cost estimation is still relevant for planning and project management as long as accuracy remains the key goal in software project development. Finally, more research should be directed to automation of testing web development to reduce the cost testing. Automation would not only help in cost reduction but also influence error management.

Reference

1. Keil P. Cost estimation for global software development [online]. Technische. Munich University: CiteseerX; 2006. URL:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.4601. Accessed 10th March, 2013.

2. Jones C. Estimating software costs: Bringing realism to estimating .New York: McGraw Hill Professional; 2007.

3. Lewis R, Wu F, and Pfleeger S. Software cost estimation and sizing methods. United States: Rand Corporation; 2005.

4. Yahya A. Latest study shows rise in project failures [online]. United Kingdom: Kinzz Business Analysis Consulting; April 2009. URL:http://kinzz.com/resources/articles/91-project-failures-rise-study-shows. Accessed 2nd April, 2013.

5. Gross G. Lawmakers look to reduce waste in US government IT contracts [online]. Framingham: CXO Media; January, 2013. URL:http://www.cio.com/article/727172/Lawmakers_Look_to_Reduce_Waste_in_US_Government_IT_Contracts. Accessed 11th March, 2013.

6. Kanaracus C. ERP software project woes continue to mount, survey says [online]. Framingham: Computerworld; February 2013. URL:http://www.computerworld.com/s/article/9236984/ERP_software_project_woes_continue_to_mount_survey_says. Accessed 11th March, 2013.

7. Evans S. Software project failures hit 5 year high [online]. London: Progressive Digital Media Group Plc; June, 2009. URL:http://www.cbronline.com/news/software_project_failures_hit_5_year_high_220609. Accessed 2nd April, 2013.

8. Madison D. Software estimating models: Three viewpoints [online]. USA: Crosstalk magazine; February 2006.
   URL: http://www.crosstalkonline.org/storage/issue-archives/2006/200602/200602-Jensen.pdf. Accessed 2nd April, 2013.

9. Birrell N, Ould M. A practical handbook for software development United Kingdom: Cambridge University Press; 1988.

10. Dorfman M, Thayer R. Software engineering: The development process University of California: USA; 2005.

11. Chen Z. Reduced-parameter modeling for cost estimation models. University of Southern California: ProQuest; 2006.

12. Conte S, Dunsmore H, Shen V. Software engineering metrics and models Michigan University: Benjamin/Cummings Pub,Co;1986.

13. Boehm B. Software engineering economics. Michigan University: Pearson Education; 1981.

14. Computer Society of India. The Journal of the computer society of India 2004; 34 (2-4):34.

15. Gibbons J. Nonparametric Statistics. New York: SAGE Publications; 1993.

16. Chandra S, Avadhani P, Abraham A. INDIA 2012 advances in intelligent and soft computing. Vishakhapatnam, India: Springer; 2012.

17. Chemuturi M. Software estimation best practices, tools, and techniques: A complete guide for software project estimators. Florida: J. Ross Publishing; 2009.

18. Sundar D. Software engineering. New Delhi: Laxmi Publications, Ltd; 2010.

19. Mishra J. Software engineering. Mumbai: Pearson Education India; 2012.

20. Naik S, Tripathy P. Software testing and quality assurance: theory and practice New Delhi. John Wiley & Sons; 2011.

21. Najadat H. Predicting software projects cost estimation based on mining historical data: Three viewpoints [online]. New York: Hindawi Publishing Corporation; January, 2012. URL:http://www.hindawi.com/isrn/se/2012/823437/. Accessed 13th February; 2013.

22. Mall R. Fundamentals of software engineering. New Delhi India: PHI Learning Pvt; 2009.

23. Jalali O. Evaluation bias in effort estimation. West Virginia University: ProQuest; 2008.

24. Wiegers K. Software requirements. Oregon: O'Reilly Media, Inc; 2009.

25. Merlo N. Constructive cost model [online]. Switzerland: Requirements Engineering Research group; December, 2002. URL:https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf. Accessed 8th June, 2013.

26. Virvou M, Nakamura T. Knowledge-based software engineering, volume 180 of frontiers in artificial intelligence and applications series. Amsterdam: IOS Press; 2008.

27. Kishore S, Naik. Software requirements and estimation. New Delhi: Tata McGraw-Hill Education; 2001.

28. Leonard B. GAO Cost estimating and assessment guide: best practices for developing and managing capital program costs. Pennsylvania: Diane Publishing; 2009.

29. Oman P & Pfleeger S. Applying software metrics. Los Alamitos, CA: John Wiley & Sons; 1997.

30. Szmuc T & Zielinski K. Software engineering: Evolution and emerging technologies. Baltimore: IOS Press; 1981.

31. Great place to work. 2013 Suomen parhaat työpaikat [online]. Finland: Great Place to Work Institute; February, 2013. URL:http://www.greatplacetowork.fi/best-companies/suomen-parhaat-tyoepaikat-listat. Accessed 2nd April, 2013.

32. Sanghera P. Fundamentals of effective program management. Florida: J Ross Publishing series; 30 Nov, 2009.