COST ESTIMATION FOR COMMERCIAL SOFTWARE DEVELOPMENT ORGANIZATIONS

by

Dinesh Tagra

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
October 2011

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "COST ESTIMATION FOR COMMERCIAL SOFTWARE DEVELOPMENT ORGANIZATIONS" by Dinesh Tagra in partial fulfilment of the requirements for the degree of Master of Computer Science.

Dated: October 21, 2011

Supervisor: _____

Co-supervisor: _____

Reader: _____

DALHOUSIE UNIVERSITY

DATE: October 21, 2011

AUTHOR: Dinesh Tagra

TITLE: COST ESTIMATION FOR COMMERCIAL SOFTWARE DEVELOPMENT ORGANIZATIONS

DEPARTMENT OR SCHOOL: FACULTY OF COMPUTER SCIENCE

DEGREE: MCS CONVOCATION: MAY YEAR: 2012

_____

Signature of Author

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The estimation of the software cost remains one of the most challenging problems in software engineering; as a preliminary estimate of cost includes many elements of uncertainty. Reliable and early estimates are difficult to obtain because of the lack of the detailed information about the future system at an early stage. However, the early estimates are really important when bidding for a contract or determining whether a project is feasible in terms of cost-benefit analysis. Estimators often rely on their past experiences for the prediction of effort for software projects. The fundamental factors that are contributing towards inaccuracy of the cost estimation process are imprecise and drifting requirements, information not readily available on past projects, and the methods that were developed and trained on specific data.

In this thesis, we have developed a software cost estimation tool that helps commercial software-development organizations to effectively and quantitatively measure and analyze the software metrics based upon the functional requirements, operational constraints and organization's capability to handle a project. This cost estimation tool is a fusion implementation or an essence of certain software measurement and estimation techniques that help a software organization to evaluate and analyze fundamental software metrics such as complexity, time, effort, and cost all of which are essential to improving turnaround time and attaining organizational maturity. The new cost estimation method is proposed for the iterative software development projects. The use case technique is implemented per iteration for the specification of the software requirements. COCOMO II and Function Point were used to compute the effort required for successive iterations. We also computed the magnitude of relative error for successive iterations. We tested the proposed method on student projects in order to illustrate its usefulness.

LIST OF ABBREVIATIONS USED

LOC	Line of Code
SLOC	Source Line of Code
FP	Function Point
SDLC	Software Development Life Cycle
COCOMO	Constuctive Cost Model
SLIM	Software Life Cycle Management
EAF	Effort Adjustment Factor
ELOC	Estimated Line of Code
UFP	Unadjusted Function Point
SRS	Software Requirement Specification
ILF	Internal Logical Files
EIF	Extrenal Interface Files
MRE	Magnitude of Relative Error

GLOSSARY OF TERMS

Line of Code (LOC)

Line of Code is a software metrics which is used to measure the size of a software program by counting the number of lines in the text of the program's source code. It is typically used to predict the amount of effort required to develop a program.

Function Point (FP)

Function Points was proposed by Allan Albrecht to help measure the functionality of the software systems. It is used to estimate the effort required for the software development.

Software Development Life Cycle (SDLC)

SDLC is a conceptual model used in project management for developing software through business needs, analysis and design, coding, testing and maintenance.

Effort Adjustment Factors (EAF)

The Effort Adjustment Factor in the effort equation is product of the effort multipliers corresponding to each of the cost drivers for the project.

Magnitude of Relative Error (MRE)

A Magnitude of Relative Error is the ratio of the deviation of actual effort and estimated effort to the actual effort.

Constructive Cost Model (COCOMO)

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation method which was proposed by Barry Boehm. The equations and parameters are used to compute the cost estimation, which has been formed on the basis of previous experience in estimation of cost software projects.

Software Life Cycle Management (SLIM)

The SLIM is an empirical software effort estimation model proposed by Lawrence H. Putnam that describes the time and effort required to finish a software project.

Internal Logical Files (ILF's)

A user identifiable group of logically related data that is located within the applications boundary and is maintained by the external inputs.

External Interface Files (EIF's)

A user identifiable group of logically related data that is located outside the application boundary and is maintained by another application.

ACKNOWLEDGEMENTS

With a deep sense of gratitude, I wish to express my sincere thanks to my supervisor, Dr. Morven Gentleman, for this immense help in planning and supporting my research project. I am also very thankful to my co-supervisor Dr. Peter Bodorik for his help in completing the thesis. Their perpetual energy and enthusiasm in the research motivated all their advisees, including me. Their encouragement, supervision and support from the preliminary to the concluding stages enabled me to develop a deep understanding of the subject. As well, their constant encouragement, feedback and constructive comments helped improve the quality of this research immensely, making my work on this project a smooth and rewarding experience.

I would like to thank Dr. Denis Riordan for giving his valuable time to be the external examiner for my thesis defense.

I am deeply indebted to my parents and friends for their inspiration and ever encouraging moral support, which enabled me to pursue my studies.

I am also thankful to the entire faculty and staff members of Computer Science of their direct and indirect help and cooperation which made my stay at Dalhousie University memorable.

I would like to thanks to staff of the Writing Centre, Dalhousie University, who helped me a lot to correct my grammatical mistakes, citations as well as to make thesis more structured. Last but not least, I offer my most sincere thanks to everyone who supported me during the course of this thesis work.

CHAPTER 1 INTRODUCTION

1.1 Cost Estimation

After some 40 years of research, estimation of the software -development cost remains one of the most challenging problems in the software engineering. Estimating the costs has troubled system analysts, project managers, and software engineers for decades. An estimate of cost and schedule is based on the prediction of the size of a future system. Unfortunately the software profession is notoriously inaccurate when estimating cost and schedule [3] [6].

Preliminary estimates of cost always include many elements of uncertainty [1]. Reliable early estimates are difficult to obtain because of the lack of the detailed information about the future system at an early stage. However, early estimates are required when bidding for a contract or determining whether a project is feasible in terms of a cost-benefit analysis. Since the process prediction always guides decision making, a prediction is always useful if it is reasonably accurate.

Many cost estimation methods and tools are difficult to use and to interpret to be much help in the cost estimation process. Several studies were conducted to evaluate the cost models. Research has shown that the estimation accuracy is improved if models are calibrated to a specific organization. Estimators often rely on their past experience when predicting effort for software projects. It is therefore of a vital need for the software industry to develop new cost estimation methods that are easy to understand, calibrate and use.

The first approach [1] that comes in to existence when talking about software cost estimation is the parametric model, which was known in the 1980's and before as Barry Boehm's Constructive Cost Model (COCOMO), and Larry Putnam's Software Life Cycle Management (SLIM). Traditional cost estimation models take software size as an input parameter, and then apply a set of adjustment factors or cost drivers to compute an estimate of total amount of effort required for the software project. The estimation at the various points is often error-prone, tedious, or just impossible to perform early in a project. Often, the only possible approach is to measure the size of the project or estimate

its cost based on expert judgments. In the late 1960's, the Delphi technique was proposed, in which a small number of experienced people guess the size of the project based on already completed projects in the organization. Research statistics suggested that most of the researchers who were working with the software cost estimation came up with the same difficulties, i.e., when the software grows in size and complexity than it is very difficult to predict and estimate the cost for software projects. Still there is a lack of rules and standards for the available software cost estimation methods which are the barrier to improving the overall software cost estimation process in the software industry.

1.2 Research Problem

The software organizations have handled many projects from various business verticals and every project was a learning experience. Specifically, for each project, an organization relied on the past experience or 'instinct' in giving the final commitment to the customer about the project's feasibility, time, and cost. However, this is rarely simply an upfront activity. Up-front estimation of the total effort is always incorrect. Many costs are only revealed during development, costs such as workarounds for faulty tools, invention of new algorithms and data structures when the preselected choices turn out to be inadequate, redesign to overcome performance inadequacies, and ramp-up costs for unfamiliar new technology are typical.

The intrinsic problem with the software cost estimation is due to the inaccuracy of the software cost estimation models that fit the different software development environments. The other factors that are contributing towards the inaccuracy of cost estimation, such as, imprecise and drifting requirements, information not readily available on past projects, and the methods that were developed and trained on specific data, do not easily transfer to other environments [1] [3] [6].

Software projects vary over an enormous range, from a one person project costing a few thousand dollars to megaprojects involving thousands of people and costing hundreds of millions of dollars. Any plausible method or tool must cope with this range. One consideration is the deviations that are expected from the predicted cost. It is clearly unreasonable to expect the estimate for a $1000 project to have the same absolute

accuracy as the estimated cost for a $100 million project. Some cost estimation models, however, could hope to achieve the same relative accuracy, i.e., the expected deviations might be proportional to the predicted cost.

The other critique towards the cost estimation is the user communication which consists of the factors regarding the customer and their static requirement in nature. This is usually the most important factor that is responsible for inaccurate cost estimation. The requirements are always volatile in nature and iterative software development recognizes this and attempts to accommodate for it. In the beginning of an iterative software development, there is incomplete and unclear set of requirements that leads to a difficulty in accurate cost estimation, but this becomes less of a factor with subsequent iterations. It is unclear what appropriate factors need to be considered for the iterative software-development cost estimation [34].

Simple software cost analysis methods are available but they are not always safe to use. The simplest method is based on the cost and productivity rates of previous projects. This approach is suitable if a new project doesn't have any cost critical differences from those previous projects, but it is not always safe if some critical cost drivers are ignored [31]. Any cost driver explanation can produce a formula for cost in terms of parameters. Given that data has been collected, those parameters can be chosen to get a best possible fit. But the relevance of the formula using best fitted parameters might be incorrect as it is subjected to the extent of deviation of the observed data from the fitted formula (using the optimal values of the parameters). There is still a question to think about the deviation between the prediction of the formula and the observed data. That is the serious flaw in the published results.

1.3 Objectives

The first objective of this thesis is to develop a cost estimation tool that helps commercial software development organizations to effectively and quantitatively measure and analyze software metrics based upon the Functional Requirements, Operational Constraints, and Organization's Capability to handle a project. Requirements engineering is a major part of Software Engineering, and every general reference on Software

Engineering (e.g. Pressman and Summerfield) emphasizes the topic. Function Points, especially as defined by the International Function Points User Group (IFPUG), are all about (and only about) cost and effort estimation based on functional requirements. Different software organizations attempting to implement the same software specification will have widely different experiences, based on the chosen software process and the organization capability based on skills and experience with the process and software projects, which impact the cost estimation.

Once the important metrics of the prospective project are determined, the organization can safely give near-precise commitments about the Feasibility and Time and Cost involved for that project, which helps in project's maturity and increasing the credibility. In this tool, important cost drivers are included for the improvement in the cost estimation process. Our cost estimation tool will assist commercial software development organizations in the following ways:

- Determination of project complexity on the basis of its features, required team-size, and available timeline.
- Determination of project size in terms of lines of code on the basis of Functional Requirements, Operational Constraints, and other significant cost parameters of the project.
- Estimation of the total effort in terms of person-months.
- Estimation of the total time required for the development of project.
- Estimation of the head count or manpower required in the project, based on the project's size and complexity.
- Analyses and comparisons with the results of estimation for various projects of varying size and complexity that were done in the past.

The other major objective of this research is to propose a method for improving the cost estimation process for the iterative software development projects. There is always an

incremental code change, which is the essence of iterative, component-based development or of maintenance. Not just agility, but incremental delivery and evolutionary (multi-release) deployment are often essential. Existing software cost estimation models and methods do not apply to this – they are for complete projects starting from nothing. There is an interesting challenge to improve the cost estimation process for the iterative software development projects.

1.4 Importance of Accurate Cost Estimation

Accurate cost estimation in the software industries is really important due to the following reasons [28]:

- For the better management of the software development projects, the needs of the resources should be completely matched with the real needs or the requirements.
- Customer always expects that the estimated software development cost should be matched with the estimated software development cost.
- The overall business plan of a software organization can be improved with accurate cost estimation, as it will lead to an efficient use of the resources.
- Cost estimation process is used to decide which resources are required for the project and how to better utilize resources.
- The accurate cost estimation process is necessary for defining the resources needed to produce, verify and validate the software products and for managing the software development activities. It also helps to decide whether cost of tools is offset by improved productivity.

1.5 Outline

This Chapter discussed the cost estimation and presented the objectives of the thesis. Chapter 2 provides a detailed background on the cost estimation and also various software cost estimation methods. Chapter 3 explains the proposed software cost

estimation tool for the software industries, while using screenshots for explanations. The proposed software cost estimation method for iterative software development projects and the related experimental results are described in Chapter 4. Chapter 5 offers conclusions and future work on the proposed cost estimation tool and method.

CHAPTER 2 BACKGROUND AND LITERATURE SURVEY

2.1 Literature Review

Software cost estimation has been a particularly active research area, with the research increasing substantially over the past two decades. For instance, surveys conducted in [7] and [8] show that research in last 25 years focused on different levels of software estimation.

Some of the software estimations were conducted at project levels and some at company levels. However, most of the literature concentrated on how commercial-level software organizations estimate project costs and the importance of the required accuracy of effort for the projects.

Surveys conducted by Jenkins [9], Phan [10], Bergeron and St-Arnaud [11], Heemstra and Kusters [12], Lederer and Prasad [13] [14], and Sauer and Cuthbertson [15] investigated the average estimation accuracy and frequency of overruns. The summaries of these surveys are displayed in Table 2.1 below. The x in the table indicates that this information was not presented in the surveys.

Study(first author)	Jenkins	Phan	Heemstra	Lederer	Bergeron	Sauer
Study year	1984	1988	1989	1991	1992	2003
Cost Overrun	34%	33%	x	x	33%	18%
Project used more than estimated effort	61%	x	70%	63%	x	59%
Project used less than estimated effort	10%	x	x	14%	x	15%

Schedule Overeun	22%	x	x	x	x	23%
Project Completed after schedule	65%	x	80%	x	x	35%
Project Completed before schedule	4%	x	x	x	x	3%

Table 2.1: Previous Surveys on estimation accuracy [7]

The studies suggest [7] that a large number of projects (60-80%) are completed with under-estimated effort and under-estimated scheduling. Sauer and Cuthbertson [15] came up with lower number of magnitudes of project and schedule overruns because study was affected by self-selecting samples of projects.

The study performed by Prasad and Lederer [13][14] was at a project level. They concluded that 87% of the companies estimate large projects only. Moores and Edwards [11] concluded in their study that 91% of the managers reported that the level of estimation accuracy was typically less than 20%.

Phan [10] surveyed the overruns at organization level and discovered findings that were incompatible with other surveys. Of the 191 organizations he surveyed, he concluded from the viewpoint of a different measurement scale, that cost overruns always occurred in 5%, usually in 37%, sometimes in 42%, rarely in 12% and never in 4%. From a schedule overrun point of view, he concluded that schedule overruns occurred always in 1%, usually in 31%, sometimes in 50%, rarely in 15%, and never in 3% of the organizations.

Pressman [17] suggests that the Buy vs. Make decision should be considered in determining the software estimation. Reusable software components are widely used by many commercial software development organizations. Project managers frequently use

the commercial off-the-shelf (COTS) components instead of encouraging developers to create from scratch. Therefore, buy/estimate decision should be an important factor in software estimation. There should be a need for the ability to predict the cost of using COTS components [35].

Parkinson's Law [16] suggests that the project cost is highly dependent on whatever resources are available in the organization, and that the project life cycle is expandable to meet the deadline according to the resources available to the organization.

Pandian [18] proposed three estimation methodologies, which he described as Analogy method, Top down Method and Bottom up method. The Analogy method involves estimating the project by using historical data of previously completed projects. The project to be estimated is compared with already existing information on completed projects. The top down approach concentrates on the overall characteristics instead of the functional and non-functional requirements of the system to be developed. The bottom up method provides the most detailed estimation because it considers each and every component and then combines them all to give the overall required estimation for the project.

McConnell [19] stated that numerous surveys have found that more than half of the projects substantially overrun their estimates and a large number of projects either cancelled or misses its delivery dates. The effective software estimation is one of the most important and difficult software development activities. The over-estimating and the under-estimating of a project are both bad for different reasons. An underestimating of a project will lead to under staffing, under scoping the quality assurance effort, short schedule, etc., whereas overestimating will cause for a project to take at least as long as it was estimated [3]. In the following table 2.2, Boehm [5] listed some major projects that cancelled due to estimation failure.

Project	First Cost ($M)	Last Estimate Cost($M)	First Schedule (months)	Last Estimate Schedule (months)	Status at Completion
PROMS (Royalty Collection)	12	21+	22	46	Cancelled, Month 28
London Ambulance	1.5	6+	7	17+	Cancelled, Month 17
London Stock Exchange	60-75	150	19	70	Cancelled, Month 36
Confirm (Travel Reservation)	56	160+	45	60+	Cancelled, Month 48
Master Net (Banking)	22	80+	9	48+	Cancelled, Month 48

Table 2.2: Software Overrun Case Studies from Boehm [5]

The Literature suggests several reasons for the overruns of the cost estimation; the factors as listed by Linda L. Laird [3] include the lack of education and training, confusion of the desired schedule/effort target with the estimate, and incomplete, changing, and creeping requirements. Many people in the software industry don't know how to estimate costs, have no training in the area of software estimation, and receive no feedback to improve the estimation process. The other reasons identified by Laird for project overruns included incomplete and unclear requirements, difficulty managing the schedule of the project as the scope change, planning an overly assertive schedule, and insufficient resources for the project. Galorath and Evans [4] conducted an intensive research by using 2100 internet websites and came up with several reasons for the failures of the software projects. The most common and important reasons he found were the insufficient requirement engineering, poor planning, sudden decision at the early stage at the project, and inaccurate cost estimation. Boehm suggested three basic reasons for inaccurate cost estimations, i.e., lack of clear understanding of the software requirements, under-estimation of software size, and required effort for the software projects.

Software Cost Estimation is an important, but a difficult, task since the beginning of the computer era in the 1940s. In the last 3 decades, there have been three models that have been significantly used for cost estimation namely: Boehm's COCOMO [20], Putman's SLIM [21], and Albrecht's function point [22]. Most of the models use the size measurement methods such as Line of Code (LOC) and Function Point (FP) for determining the cost estimation. The accuracy of the cost estimation is directly related with the estimation of size.

Boehm [6] commented that there are large numbers of cost analysis methods available, but they are not always safe to use. The simplest method is to base a cost estimate on the typical costs or productivity rates of previous projects. Some of the simple methods are useful if the new project does not have any cost critical differences from the previous projects. However, they are risky if the critical factor of the cost driver has been discarded.

Boehm [6] is known as the leader of the software cost estimation and reformulated his model in COCOMO II in 1997 which consists of three different sub models: application composition, early design, and post-architecture.

The software maintenance is also important because it consumes a large part of the overall cost of the life-cycle. The survey [37] suggested that around 75% of the maintenance effort was due to the adaptive and perfective maintenance.

In the Late 1970's different models were proposed, such as SLIM [24], Checkpoint [25], Price-S [26], SEER [26], and COCOMO [27]. Most of the researchers, who were working on developing the cost estimation, found the same difficulties when software grows in size and complexity, making it very difficult to predict the cost of software development.

Some authors suggested that the big challenge in the software-cost estimation is that there is a lack of specific rules and standards to control the overall process of software development and there is a need of some kind of rules and standards to control and improve the cost estimation process.

The inaccuracy of cost estimation is to recognize the three related quantities, i.e., functional specification, cost, and delivery time. These three quantities lead to a different form of a contract: fixed price or fixed delivery date; but only functionality can be met within that constraint.

2.2 Cost Estimation Techniques

Several methods are available in literature for cost estimation. Basically, cost-estimation methods are categorized into two groups, which are algorithmic and non-algorithmic. In this section, I am going to discuss the methods which are used for cost estimation [1] [4] [29].

2.2.1 Algorithmic Methods

The algorithmic methods use special algorithms to perform the cost estimation. The data is required to compute the results by using mathematical relations. Currently, many software estimation models are using these methods for cost estimation. The generic representation of the equation is as follows:

Effort = f(x1,x2…,xN),

Where, (x1…xN) is the vector form of the cost factors.

Most of the models are using product factors, computer factors, personnel factors, and the project factors.

2.2.1.1 Putnam's Model

This model has been proposed by the Putnam's by the inspection of several software projects and distribution of the manpower. The Software equation suggested by the Putnam's model is [4]:

$$S = E(Effort)^{1/3} Td^{4/3}$$

Where,

E= environment indicator that reflects the development capability, which can be derived from the historical data using the software equation

Td= time of delivery

Effort is represented in person-year and S is expressed in Lines of Code (LOC). The Effort equation for the model is

$$\text{Effort} = D * Td^3$$

Where,

D = manpower build-up factor

SLIM (software life cycle management) is a tool that works according to the Putnam's model.

2.2.1.2 Seer-Sem

SEER-SEM model has been proposed by Galorath Inc. in 1980. The parameters suggested by this model are for commercial and business projects. The effective size of the software project is the most vital feature in this method and is represented by Se. The effective size is calculated by determining five indicators which are new size, existing size, redesign, re-implementing and retesting. The generic formula for Se is [4] [29]:

Se = newsize+existingsize(0.4Redesign+0.25reimp+0.35 retest)

After determining the effective size, the estimated effort is calculated as:

Effort =Td = $(D)^{-0.2} * (\frac{Se}{Cte})^{0.4}$

Where D = degree of staffing complexity

Se=effective size

Cte = productivity and efficiency of the used development method

2.2.1.3 Linear Models

The linear models are simple in structure and can be expressed by the following equation [4]:

Effort $= a(0) + \sum_{i=1}^{n} \text{ai} * \text{xi}$

Where a1,….an are selected according to the information available for the projects.

2.2.1.4 Constructive Cost Model

The Constructive Cost Model (COCOMO) is proposed by Barry Boehm in 1981 and is the most popular model in the algorithmic methods for the cost estimation [1] [4] [29]. The equations and parameters are used to compute the cost estimation, which has been

formed on the basis of previous experience in estimation of cost software projects. COCOMO II is the most recent version of the COCOMO that predicts the amount of effort required for the projects in person-month. The usage of COCOMO II is high and usually it produces more accurate results compared to other algorithmic methods.

The COCOMO II [38] model makes its estimate of the required effort based on the estimate of the software project's size:

Effort = $\mathbf{2.94} * \boldsymbol{EAF} * (\boldsymbol{KSLOC})^{\boldsymbol{E}}$

Where,

EAF = Effort adjustment factor derived from the cost drivers

KSLOC = Project Size measured in thousand source line of codes

E = Exponent derived from the five scale drivers

The COCOMO II [38] predicts the number of month required to complete software project. The duration of the project is dependent on the prediction of the effort:

Duration = $\mathbf{3.67} * (\boldsymbol{Effort})^{\boldsymbol{SE}}$

Where,

Effort = Effort computed from the COCOMO II effort equation

SE = Schedule exponent derived from the five scale drivers

2.2.2 Non-Algorithmic Methods

The non-algorithmic methods are based on the comparisons with the previous projects that are similar to the project being estimated. The cost estimation for these methods can be done by analysis of the already existing datasets for the projects. The non-algorithmic methods have two categories which are analogy- based and expert judgment [1] [4] [29].

2.2.2.1 Analogy based

Numerous similar completed projects are analyzed. Based on these analyzed completed projects, estimation of under-estimated project is done according to actual effort and cost. This method can be performed at system as well as subsystem levels. The steps for this method are as follows [4]:

a) Selection of analogy
b) Exploring the similarities and differences among the projects
c) Inspecting the quality of the analogy
d) Determining the estimation

2.2.2.2 Expert based judgments

As the name suggests the judgments of one or more experts are involved during the cost estimation process [4]. The expert should have extensive experience by having handled similar kind of projects in past. This method is applicable where it is hard to find appropriate data and gather the requirements. The Delphi method is the most common method, which is based upon the expert based judgments, and the steps for the Delphi techniques are as follows [29]:

a) The coordinator gives an estimation form to each expert to record the estimation.
b) The form is filled by experts without any discussion with each other.
c) The summary of all estimations are prepared by the coordinator who then requests iteration for the estimation.
d) Steps (b, c) are repeated for many rounds before finalizing the estimation.

2.3 Reasons for failure of the cost estimation models

Tom DeFanted [30] asserted in 1970's that the cost and effort estimation was a solved problem-clearly not. Today, several methods are available for the cost estimation but unfortunately none of them can estimate the cost of software with a high degree of accuracy. Several reasons are identified as being responsible for making the cost estimation processes difficult [3] [29]:

- The software development environments are evolving quickly and endlessly.
- There are several interrelated factors which affect the cost estimation process in the software development such as volatility of the system requirements, number of user screens, and the reusability of the components.
- There is still a lack of measuring the complexity of the software projects.

- Presently, there is still a lack of accurate historical data for the measurement of the cost.
- Sometimes, there is a lot of information about the past projects that is required for cost estimation, which is not available in every situation.
- Incomplete, changing, and creeping requirements are the other sources which are difficult to manage in a fixed-price and the fixed-schedule project.
- There are still a large number of factors missing which should be considerable during the process of cost estimation.

Today, there is a strong need to improve the performance of existing methods and also for bringing in new methods for the cost estimation.

2.4 Classification of Software Metrics

Software metrics may be broadly classified into three different categories: Process Metrics, Project Metrics, and Product Metrics [33].

2.4.1 Process Metrics

Process metrics are primarily used for software development and maintenance and classified as private and public metrics. Private metrics are used to assess individual team member productivity, while public metrics help to evaluate the organization as a whole and evaluate the performance and productivity of a process.

2.4.2 Project Metrics

Project metrics are used to estimate the cost and effort for software projects and includes metrics such as lines of code, cyclomatic complexity, and code coverage. In the lines of code metric, all physical (comment and blank lines) and logical (statements) lines are counted for a specific programming language. Cyclomatic complexity measures the complexity of a program or application and is computed by using a flow graph. Code coverage determines the degree to which the source code has been tested.

2.4.3 Product Metrics

These metrics focus on the key characteristics of the software product. Commonly used product metrics are specification quality metrics, architectural metrics, length metrics and testing effectiveness metrics. Specification quality metrics measure of the completeness of the requirements. Architectural metrics provide information regarding the quality of the architectural design of the system. Length metrics measure the system size by using lines of code during implementation phase of the project. Testing effectiveness metrics measure the effectiveness of executed test cases.

2.5 Characteristics of Good Software Measures

Good metrics [32] should contribute to the development models that are capable of predicting software product and processes. The metrics and models can be used to measure the productivity and product quality by estimating the product cost and schedule. The ideal metrics should have the following characteristics:

- Be simple and definable so that they will be easy to understand and evaluate.
- Have a well-defined goal.
- Be available at a reasonable cost.
- Be valid - Metrics should measure what they are intended to measure.
- Be robust - Metrics should be relatively insensitive to insignificant changes in the process or product.
- Be consistent in terms of units and dimensions.
- Be independent of programming languages.
- Give useful feedback.
- Be practical.

2.6 Size Estimation

Estimation of project size is fundamental to estimating the effort and time required to complete the planned software project. The size of a program indicates the development complexity. There are two important metrics to estimate size [3] [29] [39].

- Line of Code (LOC)
- Function Point (FP)

2.6.1 Lines of Code

The software industry began in the 1950s, with the early metrics being SLOC (source lines of code) [39]. Line of code and source line of code have the same meaning, and this size metrics is widely used in the software industries. Source line of code is defined as measuring the size of a software program by counting the number of lines in a program [3]. The Lines of code or Source Line of Codes is based either on physical lines of code or logical lines of code.

The main strong points of physical lines of code (LOC) are
i) This is really an easy measure for counting the size of application.
ii) We can easily automate the measurement of physical line of code.
iii) The physical line of code can be used by software estimation tools.

The main strong points for logical statements are
i) The logical statement cannot consider dead code, blank lines and commented lines.
ii) The mathematical conversion of logical statement to function point is possible.
iii) The estimation tools can be applied to logical statements.

2.6.1.1 Limitation of LOC

- A programmer whose productivity is measured in terms of LOC will try to write unnecessary verbose code. This can increase the complexity of the system and also increase the effort required for bug fixing.

- The estimation of LOC count at the beginning of the project is very complicated compared to estimation at the end of a project. In measuring the LOC at the beginning, the problem should be divided into different modules and each module should be divided into sub modules to predict the project's size and complexity.

- The greatest emphasis of LOC is on measuring the coding activity. Thus, it usually ignores the complexity of design and testing activities. Design may be equally as complex as coding activities.

- Some programmers write extra amounts of code to make the code structure complex in order to increase productivity in terms of line of code. This type of unnecessary code creates extra overhead and is not an adequate measure for size estimation.

- Code reusability also affects the estimated efforts in terms of Line of Code. Reusable amounts of code should be excluded from software metric estimation. Reusability, even just in the sense of exploiting libraries, reduces Lines of Code. More suitable domain-specific languages also reduce Line of Code [29] [39].

2.6.2 Function Point Metrics

In the 1970s, Allan Albrecht came up with the Function Point (FP) method for size estimation, which can be applied to different programming languages or to a combination of programming languages. The Function Point is computed from the analysis of the requirements and specification of the application. It consists of five weighted and adjusted factors [36] [39]:

1. Number of Inputs (screens, signals, etc.)
2. Number of Outputs (screens, reports, checks, etc.)
3. Number of Inquiries
4. Number of Logical files
5. Number of Interfaces

The main strong points for function point metrics are [39]:

1. Function points always remain independent of programming language used.
2. Function points perform well for the analysis of complete software life cycle.
3. Function points are able to measure coding as well as non-coding activities like documentation.
4. Function-point analysis is also able to measure defects in requirements and design.
5. Function-point analysis can be used by software estimation tools.

2.7 Iterative Software Development

Iterative software development is at the heart of the software development process and it was developed to overcome the weaknesses of waterfall model [34]. The iterative software development accommodates the volatility of the requirements. The common feature of the most iterative methods is that after each iteration, a running product is developed. A final clean-up may be required before ending the project, but a well-managed iterative process allows the project to be declared at any stage. Within each iteration, there will be some requirements analysis to choose what this iteration is intended to add to (or remove from) what was produced in the previous iteration. This might be done partly by the customer and partly by the software development organization.

2.8 Metrics for Planning and Controlling the Projects

The other flaw to cost estimation observed in the literature is that an organization cannot estimate the software development cost accurately because it does not have the measurements available from the previous projects. Metrics of previous projects are useful for planning and control of the future projects. Putnam and Myers suggested some useful metrics for the planning and controlling of the future projects [1] [30]:

- Amount of function built or modified usually is measured in terms of lines of code and often function point, subsystem and use cases.
- Development time is required to number the months or years to complete the implementation stage of the project.
- Effort applied is measured by the number of person–months. It is also useful to estimate the cost of the project.

  Cost = person-months × average labor rate
- Process productivity is represented as a rate at which the work is accomplished.
- Defect rate measures the quality and reliability of the project.

CHAPTER 3 PROPOSED COST ESTIMATION TOOL

The proposed software cost estimation tool will help commercial software development organizations to improve, strengthen, and add value to their business processes through reducing response times and bringing accuracy in their answer(s) to a particular business requirement. The two estimation models (Basic and Intermediate) have been implemented based on the methodology of gathering inputs regarding a certain business software project. In performing the cost estimation, our tool considers parameters such as Functional Specifications, Operational Constraints, and Organizational Maturity to handle a particular business requirement or project. This tool is a fusion implementation or an essence of certain software measurement and estimation techniques that help a software organization to evaluate and analyze fundamental software metrics such as Effort, Time, People and Cost, all of which are essential to improving turnaround time and attaining organizational maturity. Cost estimation tool is actually, or primarily, a compostion of COCOMO and Function Points. Versatility of the proposed tool is the function point method that is used to measure the project size in terms of lines of code and also the COCOMO (Basic and Intermediate) are implemented for the different levels of the complexity of the project.

3.1 Complexity of the Project

Whenever a software organization wins a new project, the first and perhaps the most important step during initial discussions with the client in the requirements-gathering phase is to determine the complexity of the project.

3.1.1 Functional Requirements Complexity Determination

The determination of the complexity is significant in order to evaluate the total effort and time for its implementation. There is no universal consensus on how to define, characterize, or measure complexity. Theoretical computer scientists tend to assess algorithm complexity by counting the number of steps taken by the algorithm for a problem of a given size performed on a particular model of an abstract computer. This may or may not correlate with measured run time or other resource consumption on real

hardware. Asymptotic results are often quoted, but the sizes of problems solved in practice often are not in the range where asymptotic simplifications apply.

The following are important factors that need to be considered during the complexity determination of the different domains of the projects:

- The most obvious complexity factor is the complexity of algorithms and data structures used in the project. This might be because the mathematics is subtle and sophisticated, or it might be because the mathematical model of the scientific situation is delicate and sensitive. It might be because simpler algorithms would be significantly less efficient or that naïve mathematical simplifications are subject to excessive round-off error.

- It might be that sequencing, timing, or concurrency is critical, and not easily expressed in conventional programming languages that lead to a complex system.

- Another interpretation of complexity is that the data collection and the operational procedures when using the software are lengthy, tedious and error prone, so external input must be continually monitored for consistency, internal corroboration, and plausibility.

- It is conceivable that the coding was merely incompetent and clumsy, and there is a superior way to code computation. However, it is much more plausible that if the computation is programmed in a non-obvious way, there is some deep explanation for why that was done, and the desired effect is not a consequence of the formal definition of the programming language, but an artifact of the implementation. Cache behavior or NUMA (non-uniform memory access) behavior is an example, as are the consequences of the differences in performance between multi-core and multi-processor.

- File system layout and mapping of file structures to disk might have a significant impact on performance as well as resilience to data corruption, yet these are not part of the programmer's abstraction.

- Although security by obscurity is sometimes not the best way to ensure confidentiality, deliberate obfuscation can still play a useful role.

- Example of the errors in prediction of tsunami-wave height illustrate another form of complexity: the Navier-Stokes equations are theoretically sufficient to completely define liquid motion, and although there are technical difficulties in integrating these partial differential equations, there are approximations that are sufficiently accurate to be used for this purpose. However, the solutions do depend on the boundary conditions, and although today we do have depth maps of the whole Pacific basin, the detail is not of sufficient resolution for the predicted wave height to be within a factor of 2, or even a factor of 10. The amount of data required is enormous, and most of it simply hasn't been gathered, although in principal it could be. Even if it was known, organizing it for efficient access would not be possible at most plausible computation sites. So access to relevant data could be another form of complexity.

- Let us take an example of a real-time missile controller project that is won by an organization for the first time. Hence, they are not specifically aware of, or experienced, the challenges of this domain, i.e., Defense. To date, the organization has accomplished various projects in different verticals such as Biometrics, Network Security, and so on, but the Defense domain is altogether new for them. Moreover, since it involves issues of national security, the client has imposed certain crucial constraints in terms of the application's functionality, time and cost.

After this project is won by the organization, the first step is the identification and determination of project complexity during the initial requirements discussion with the client. In order to evaluate this, the organization would want to clarify certain points, such as:

- How critical and sensitive is this project in terms of its functionality?
- What will be the required team-size to develop this application?

- How rigid is the imposition of time constraints by the client?
- Does this project require any specific development or testing infrastructure?
- Does this project require any specific domain competency or knowledge?

All of the above queries indicate a need for determining the complexity of the project. The project manager should take above queries into account when determining the complexity of each project class. The project classes are described in Figure 3.1.

3.1.2 Implementation for the Complexity of the Project

Taking the above scenario into consideration, our application helps to fulfill the purpose by giving the Project Manager (who, we assume, would be supervising the application) five options to determine the project's complexity. Three factors affecting the option choice are: Functional Requirements, Operational Constraints, and Team Size. The classifications of the project's complexity were selected based upon the COCOMO model.

From the perspective of the user of the tool, it is important that the predictions of complexity represent predictions that the user will be able to make. We assume that the Project Manager will select the appropriate option based on his/her experiences of managing projects. Specifically, we will classify the project complexity into five broad categories.

1. Very Simple
2. Simple
3. Advanced
4. Semi Complicated
5. Complicated

This classification will play an important role in analyzing and measuring the different metrics involved in the implementation of the project / module. A sample screenshot of this feature is shown in Figure 3.1.

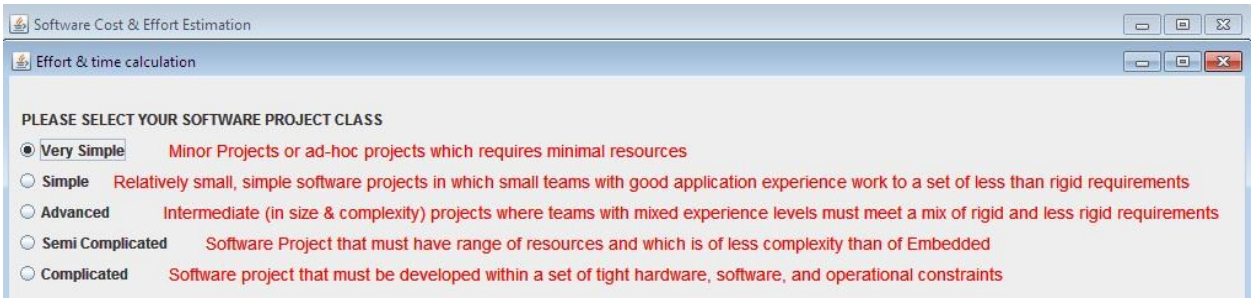

Fig 3.1: Screenshot depicting the project class/complexity level

Once the Project Manager selects the appropriate complexity, we assign a numeric value to a variable according to his/her selection of the project's class or complexity, i.e., the higher the complexity, the higher the value of the initialized variable, and vice-versa. The numeric values are assigned based on COCOMO (Basic and Intermediate) model. Please see step 3 in Appendix A for the assigned values of the complexity.

3.2 Project Size Determination

After the complexity of the project is determined, the size of the project needs to be gauged in order to use it as a metric. We assume that our application will measure the size of a project/module in terms of the Lines of Code required for its implementation.

3.2.1 Dimensions used in the project size estimation

In order to fulfill the above objective and calculate the size of implementation of a project/module, our tool enables the user to analyze it from three different dimensions, as explained below.

Dimension 1: **Functional Characteristics**

Under this dimension, the user will analyze and evaluate the functional expectations and requirements from the software that needs to be implemented for the project/module. From functional expectations, we are referring to four parameters. These parameters are taken from function point method to determine the project size based on the functionality that the system delivers to the users. The selected parameters are as delineated below:

1. User Inputs: This parameter will signify the number and level of user inputs required by the solution to be implemented.

2. User Outputs*:* This parameter will signify the number and level of user outputs required by the solution to be implemented.

3. User Enquiries: This parameter will signify the number and level of user enquiries that the solution will need to answer. The enquiries could be answered in the form of reports or some data required by the user. Specifically, it refers to the number and level of interrogations made by the user from the module.

4. Files or Databases: Refers to the number of external databases or files required by the solution.

5. External interfaces: Interfaces with the interoperability with other software.

The above functional requirements are rated according to their expectations from the necessary module that has to be implemented. This rating will actually contribute significantly in determining the size of the solution. We assume that the ratings will be chosen amongst three options, which are: Simple, Average, and Complex. This means that the higher the rating of any functional parameter, the bigger the size of the module in terms of lines of code that have to be implemented.

Hence, for example, if, in the missile-controller project, a module requires a Complex level of all of the above five functional parameters, its implementation would be bigger in size. Otherwise, it could be either Moderate/Average or Simple.

Dimension 2: **Operational Constraints**

This step would enable the user to analyze and determine the operational factors and constraints involved within the implementation of the project.

The need to determine operational constraints arises from the fact that when we develop any solution, apart from fulfilling all its functional requirements, it must also be efficient

and possess other important characteristics like reusability, low performance overhead, etc. Furthermore, when we take these factors into consideration, it becomes all the more important to develop it in such a manner that makes it feature-rich in terms of operational capabilities.

For instance, suppose, for a certain module in the project, our application will enable the Project Manager to rate the implementation of that module from many different perspectives. The rating would be done to highlight the importance of a particular operational capability within the prospective solution. This means that, on a scale of 1 to 5, if the rating for *Reusability* is 5, the required solution must be reusable. Thus, when similar classifications of modules are needed to be developed, they can be implemented immediately with some additional effort. To effectively consider the operational constraints, our application asks 14 questions from different angles and makes the user rate them, as per their relevance. These 14 questions were chosen from function point method for the consideration of the general characteristics of the project classes. The chosen questions are listed in **BLUE (**middle) rectangle of Figure 3.2.

Dimension 3: **Programming Language**

In this sub-feature, we enable the user to select the programming language in which the prospective solution would be developed. We assume that, as the generation of a programming language advances, it requires less effort during development compared to the older generation languages. Hence, if the user selects an older programming language to develop the solution, it directly affects the size. Newer programming languages should have more semantic depth, meaning that less code needs to be written because the functionality would be provided by the language and its library. A screenshot displaying the discussed three dimensions is shown below in Figure 3.2.

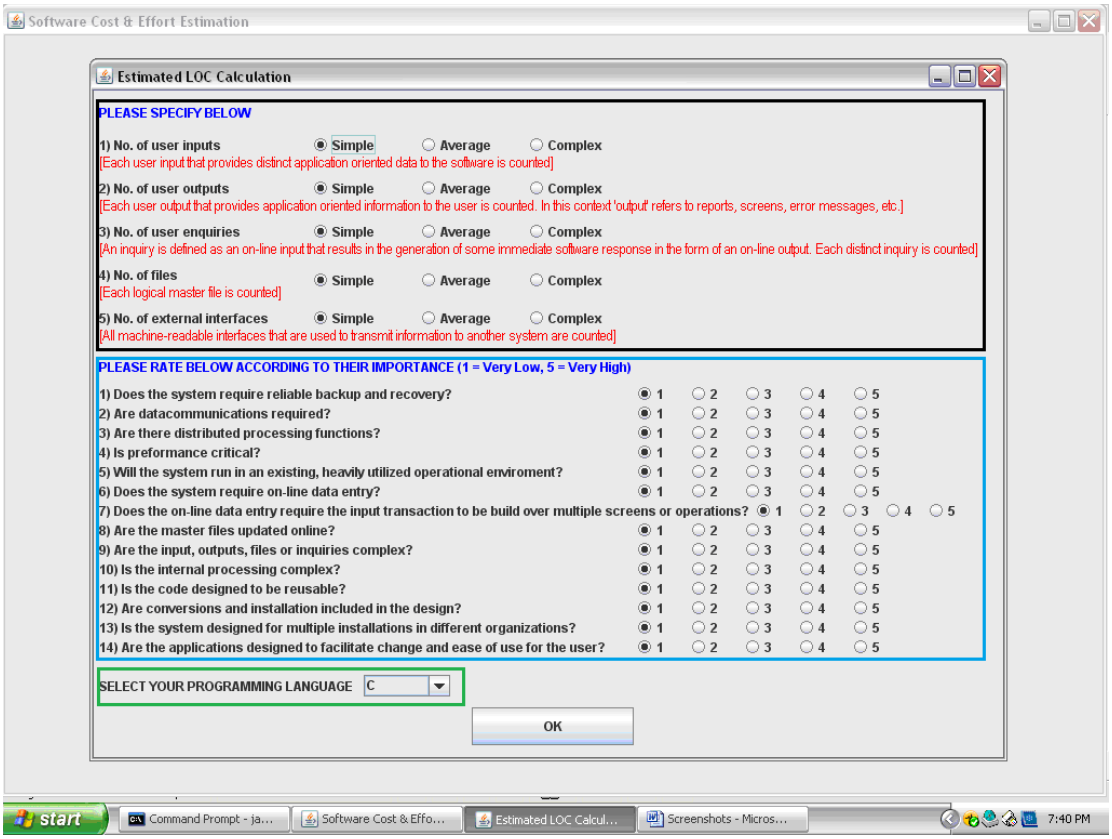

Figure 3.2: Screenshot depicting the three different dimensions

In the Figure 3.2, the three rectangles in different colors depict different dimensions, as discussed above. The **BROWN** rectangle on top signifies the functional requirements of the application or solution to be developed. It enables the user to rate the complexity and significance of functional requirements on five different parameters. The higher the rating of parameter-complexity, the larger the application size.

The **BLUE** (middle) rectangle is used to determine the complexity and importance of the operational constraints or factors. The higher the complexity, the larger the application size. The rectangle in **GREEN,** on the bottom, depicts the selection of the programming language to be used for development. The functional characteristics and operational constraints are rated for the complexity of programming languages. Functional characteristics and operational constraints rating was chosen on the basis of function point. For the rating, please see the step 4 (4.1 & 4.2) in Appendix A. We have assumed three programming languages for development, namely C, C++ and Java.

For every selection made for individual parameters above, we initialize a separate variable with a value that reflects the selected complexity level of the parameter. The selected values for the parameters were taken from function point method for their degree of complexity. Once all the selections regarding Functional Expectations, Operational Constraints and Programming Language have been made, we calculate the size. The size is computed on the basis of function point. For the computation detail of the estimated size, please see the step 5 in Appendix A.

3.3 Determining the cost factors of project

Once the complexity and size of the project are determined, the next objective is to determine the relevance of factors that drive the application's cost. These factors are important, since not only do they play a vital role in determining the total effort, time and manpower required in a project, but they can also have an impact on the total estimated cost of development. The specific calculation performed is based on the COCOMO.

Our application enables the Project Manager to determine the *Cost Drivers* through rating the importance of four attributes, as mentioned below:

- Product Attributes: These factors would highlight the core important features of the product, such as reliability, complexity, etc.

- Hardware Attributes: These attributes would reflect how much advanced hardware configurations are required to implement the solution of the business problem, project or module.

- Personnel Attributes: These attributes would help in rating the features related to human resources like efficiency, productivity, competency, etc.

- Project Attributes: These attributes would be related to the project's characteristics such as milestones, deliverables, etc.

The whole idea in rating the above factors and attributes is to have an additional input about the application or the features that are to be implemented in order to handle the project effectively. This input would further help us in effectively measuring and estimating the metrics.

Now to develop this level of capability in the missile controller software, the Project Manager needs to evaluate its implementation and sensitivity from a realistic point of view, keeping both its pros and cons in mind. In other words, the project manager must have the answers to be sure about the impact of each particular requirement (i.e., How many people would want this capability to be built-in? How would it affect the usage of the tool? How many users will benefit from this capability? And will it be of any use to other people who do not wish to use it? etc.)
After evaluating the complexity and size, an important step is to identify it in terms of other attributes which would require us to answer the questions below:

- Should an organization try to do this entire project by itself, or should we subcontract parts of the project to secondary or even tertiary suppliers with more specialized resources?
- Are the existing team members competent enough to build this feature?
- Is there any hardware required at the end-user level to implement or test this feature effectively?
- How crucial or sensitive is this project or module implementation?

The answers to the above questions will actually help us in determining the relevance of other additional factors which significantly contribute to the cost of the application. Please see the step 6 in the Appendix A for rating of the selected cost drivers.

3.3.1 Implementation of cost drivers of the projects

The screenshot of this functionality is described below in Figure 3.3. The cost drivers which we selected are based upon COCOMO.

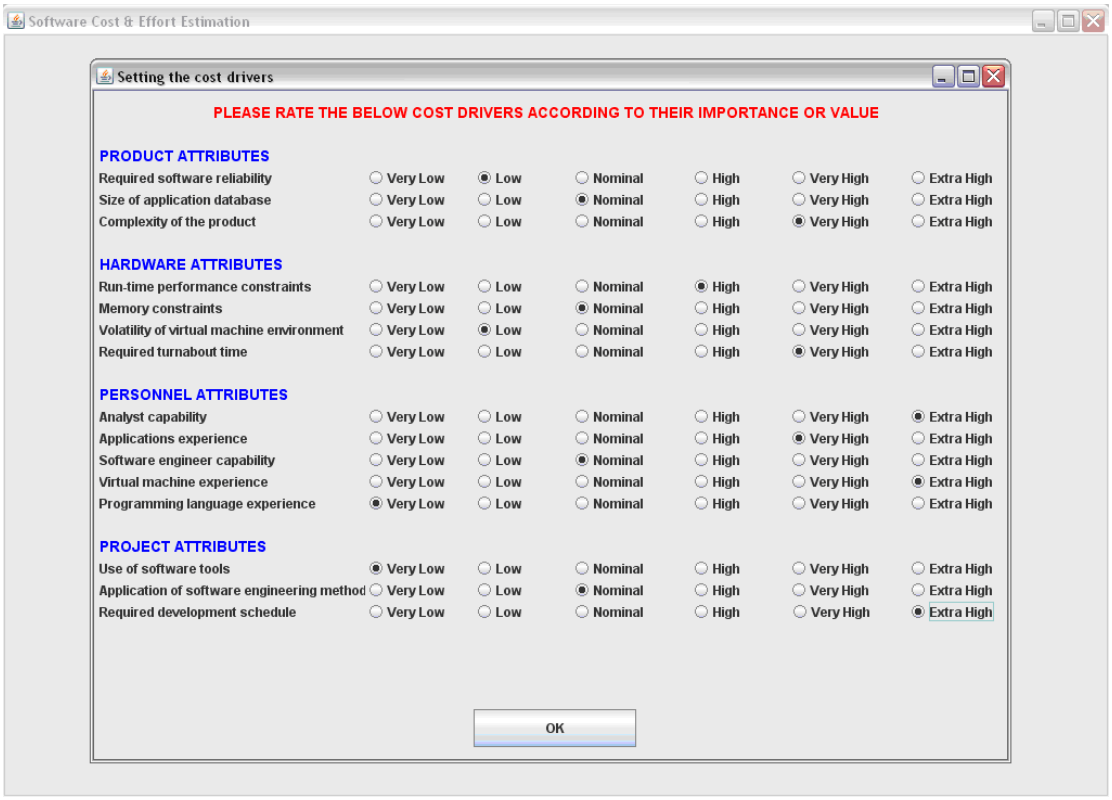

Figure 3.3: Screenshot mentioning the Cost Drivers of the application

Figure 3.3 describes how the application enables the user to rate the *Cost Drivers* according to their relevance and complexity. This means that in considering the appropriate attributes, a user rates them while taking the complexity and size into consideration. The higher the rating, the higher the estimated amount of effort, time and manpower required to develop the project or module.

At the implementation level, whenever a user puts a final rating to all the attributes after carefully understanding the functional expectations of the project / module and its corresponding cost drivers, we declare the variables and define them according to the complexity or rating level. This means, the higher the rating, the higher the variable value.

3.4 Determining estimates of Time, Effort and People

After determining the application's Complexity, Size and Cost-driving factors, we need to determine the actual metrics of the project or module that has to be implemented, i.e. we need to answer queries such as those below:

1. What is the total estimated effort required in implementing a particular project or its module?

2. How much approximate time duration will it take for the implementation?

3. How many estimated number of people would be required for the implementation?

4. What is the estimated implementation cost?

Once the above queries are answered, we can utilize the above metrics for project management and scheduling purposes.

Therefore, once we have answered the following questions, then we can move forward in determining the actual Effort, Time, People and Cost of the project's implementation. The questions which we identified are based upon functional characteristics and operational constraints to handle the project.

- How critical is this project / module?
- How useful is this project / module in terms of its merits and demerits?
- How complicated is this project / module?
- How big is it in terms of implementation size?
- What are its high and low-level functional requirements?
- What are its desired operational constraints?
- What is its feasibility analysis?

3.4.1 Implementation for Effort, Time and People

In order to calculate the total estimated Effort, Time, People and Cost required for the implementation of the project or module, we would use the inputs, complexity and size of the solution. In our application, we have planned to calculate the size in three ways:

1. By considering the functional requirements
2. By considering both functional and non-functional requirements and operational constraints
3. By considering the complexity of the project to be developed

For the implementation of effort time and people, we have implemented two estimation models of COCOMO which are Basic & Intermediate, respectively. A screenshot showing where we enable the user to select the model is highlighted in Figure 3.4.

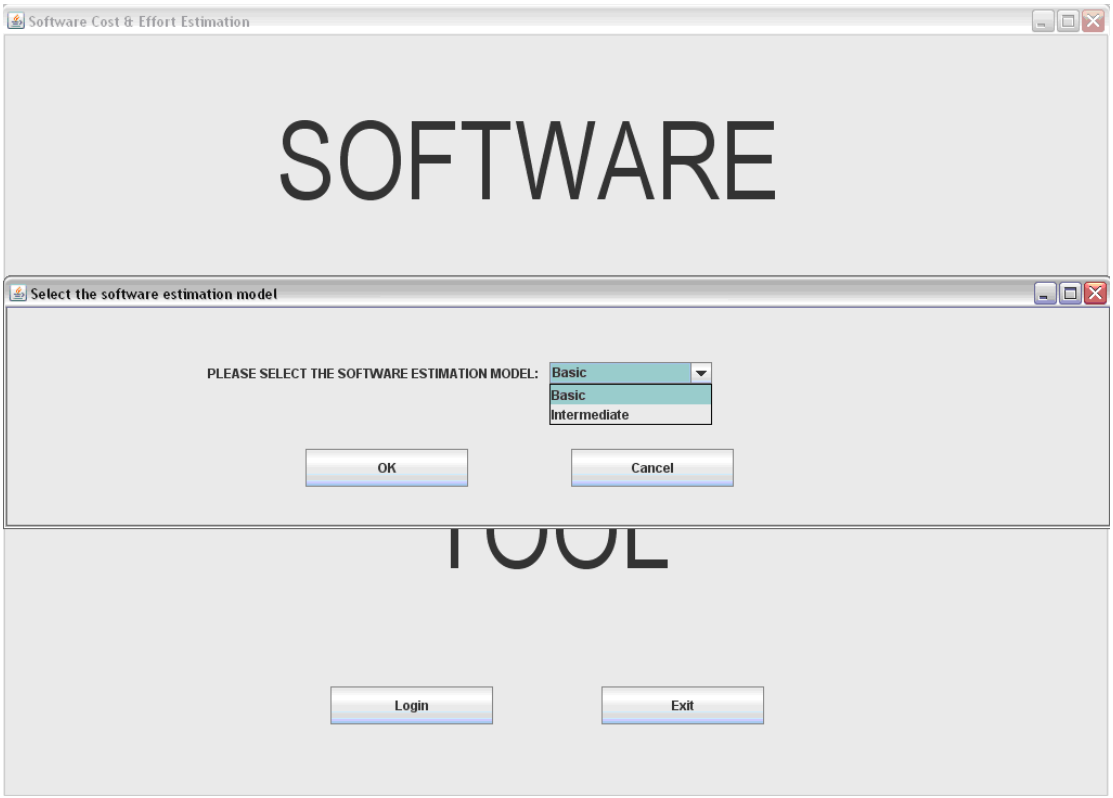

Figure 3.4: Screenshot depicting the selection of software estimation model

Once the user selects the estimation model type, as highlighted in the **RED** rectangle above, the next screen or interface will be presented accordingly. For instance, if the user selects the *Basic* model of estimation, he/she will be asked to input only the functional requirements for size-estimation. If the user selects the Intermediate model for estimation, functional requirements, operational constraints and attributes will also be taken into consideration in the form of cost drivers. A screen shot of the interface when the user selects Basic estimation model is shown in Figure 3.5.

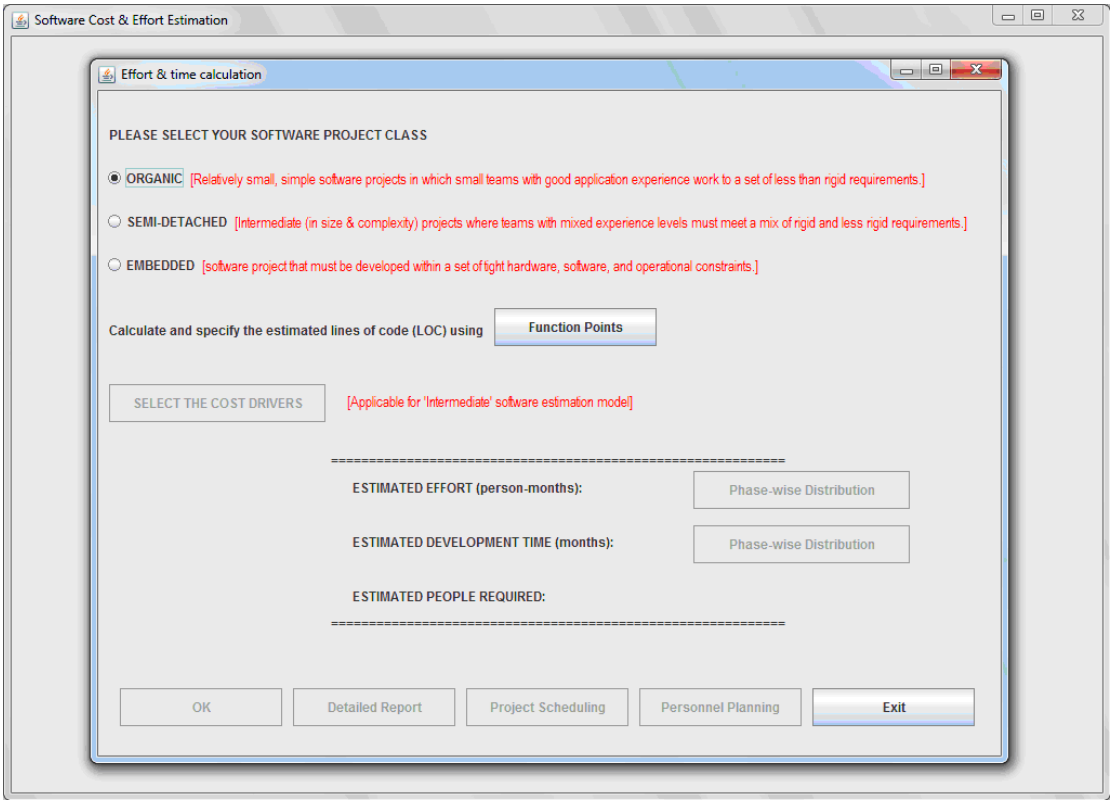

Figure 3.5: Screenshot highlighting the "Basic" estimation model

The PINK rectangle above signifies that if the user selects the *Basic* model, he/she will only be asked to supply functional requirements for size estimation. The operational constraints or Cost Drivers will remain disabled.

Once the user has performed the size evaluation for both models, he/she can better conclude the evaluation of metrics through approximation. However, the application's complexity-evaluation will be done for both the models.

Once we have evaluated the size and complexity of the prospective implementation for the project/module, we will use the size factor to determine the estimated effort and complexity factor to determine the time metric. Then, once the estimated effort and time have been calculated, the estimated required number of people in the project can be calculated by dividing the estimated Effort with Time.

Considering the above-mentioned example, let us assume that the total number of estimated people required for implementing the module is 30 and the required time is six months. In such a case, the implementation cost can be roughly calculated by adding the respective salaries of each employee multiplied by the total time for which they would work on the implementation plus other expenses and profit figures.

After processing the final inputs from the user regarding the Functional Requirements, Operational Constraints and Organization's Capability or Maturity to implement the project or module, our application would declare and define various intermediate variables in order to store the values of calculated Size, Complexity, Effort, Time, People and Cost, and would use each variable accordingly.

A screenshot displaying the values of estimated metrics is shown below in Figure 3.6. The organic, semidetached and embedded are the project classes for COCOMO which represents simple, advanced and complicated levels of complexity respectively. The specific calculation is performed based on the COCOMO (Basic and Intermediate) and the formulas are described in step 7 of Appendix A.

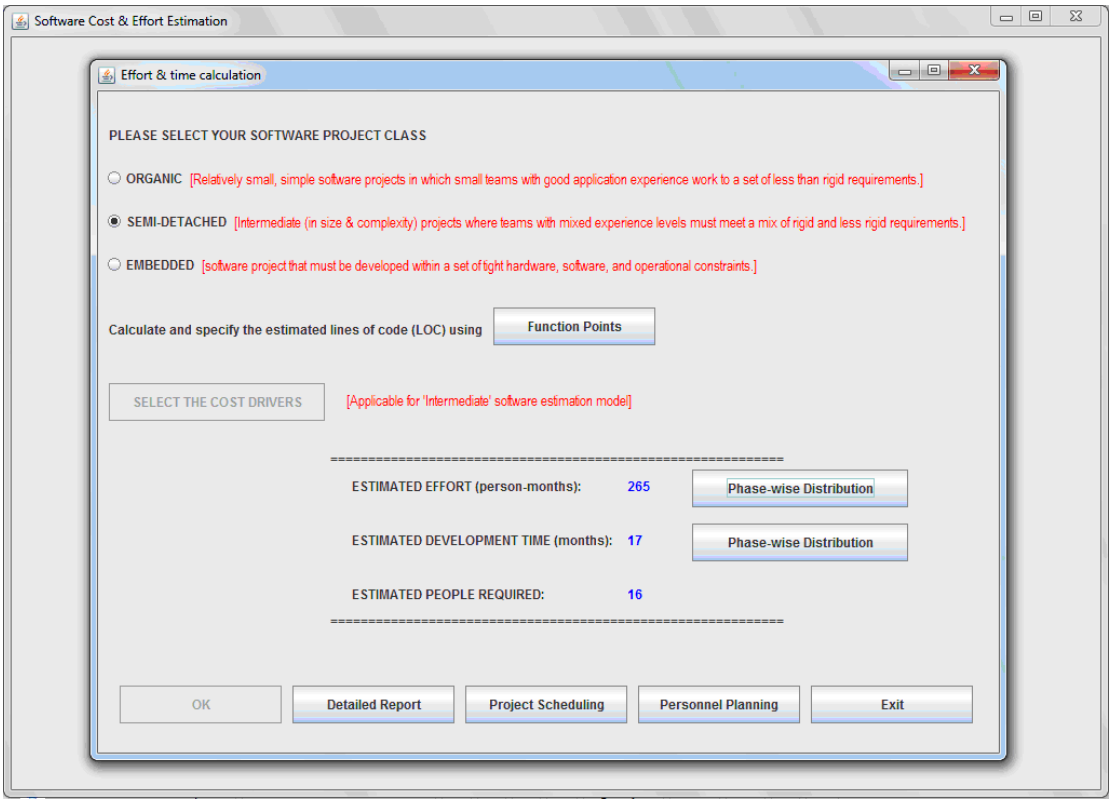

Figure 3.6: Screenshot highlighting the estimated Effort, Time & People

The rectangle in **BROWN** above highlights the calculated estimated values of the metrics, Effort, Time and People.

3.5 Recording of Estimated Data

The next most important objective of software estimation and measurement practices is the recording of estimated data. In other words, in order to save time and bring efficiency and maturity into software cost estimation process, the organization should record the estimation data for comparison and analysis purposes for future projects. For example, the organization receives a similar magnitude of requirements from the Defense domain again, but this time they need to control high-altitude projectiles or intelligent nuclear bombs dropped from air to surface. The project has been evaluated to be of a similar complexity and size as the previous one; thus, if the estimated data from the previous implementation is stored, it will not take the same amount of time as before for the organization to plan and schedule the implementation. In other words, if the appropriate records are there, the organization can fetch them and, based on that data, conveniently

and speedily manage the implementation of the new project or module without any hassles or delays, all the while taking care to incorporate into the latest project any significant facts or lessons learnt during the previous implementations.

Additionally, the recording of data also helps the organization to compare and analyze the results in order to extract meaningful inferences. These can also be adopted into their software engineering practices on their way to becoming a mature organization.

3.5.1 Implementation for the analysis of recorded data

We have implemented the recording of analysis data in the form of a detailed report that contains all the information related to the analysis of previous projects. The report includes details like Analysis Timestamp, Project Class or Complexity, Estimated Effort, Time in the duration of months and days and Number of people.
A screen shot of this information is displayed below:

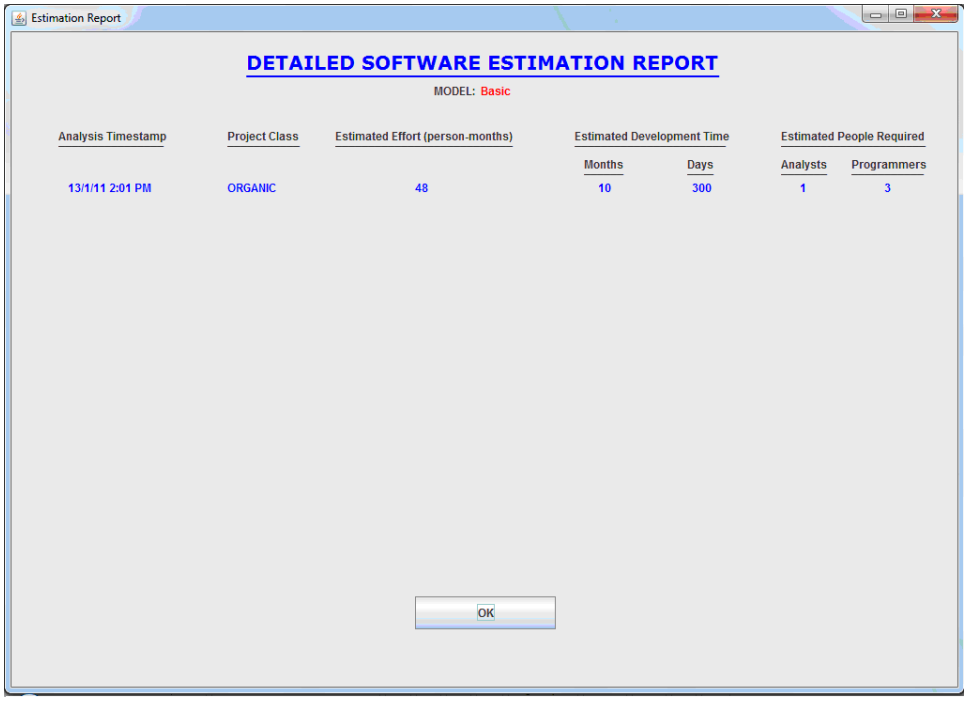

Figure 3.7: Screenshot highlighting the estimation data using “Basic” estimation model

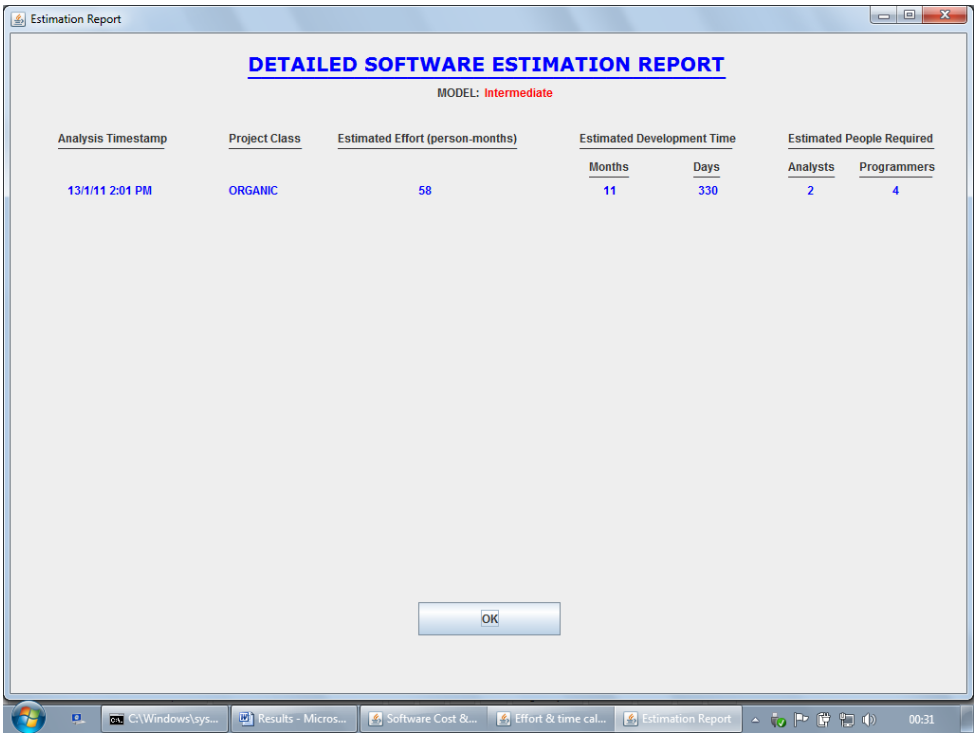

Figure 3.8: Screenshot highlighting the estimation data using "Intermediate" estimation model

As shown in the above screenshots, we can record the results of estimations from both implemented models – Basic and Intermediate – individually so that whenever there is a need for comparison or analysis, we can refer to the stored data.

3.6 Implementation of Cost Estimation Tool

The flowchart of the implemented tool is shown in Figure 3.9 (shown on the next page).

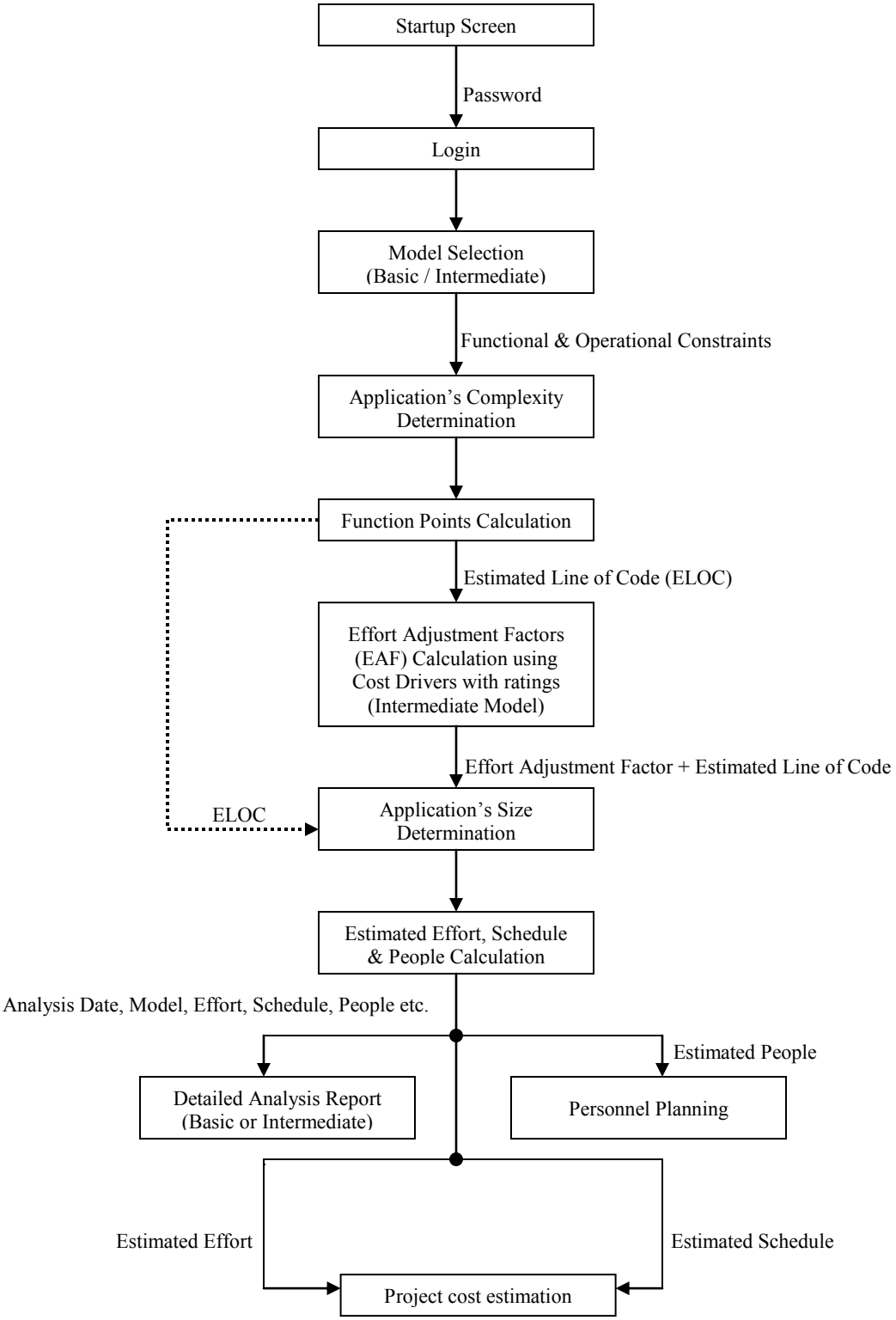

Figure 3.9: Flow Chart of proposed cost estimation tool

CHAPTER 4 COST ESTIMATION METHOD

The purpose of the proposed method is to improve the cost estimation process for iterative software development projects. The two majors steps of this method are determination of the use case required for each iteration and the computation of effort estimation in each iteration. To accommodate the requirements at each iteration, use case technique was used. The function point method is used to determine the unadjusted function point for each iteration, where the COCOMO II is used to compute the required effort in each iteration. The section 4.1 discusses the method in detail while the testing results of proposed method are discussed in section 4.3.

4.1 Effort Computation for Iterative Software Development Projects

The two main steps of the method are required in each iteration:

Step 1: Identification of Use Case required
Step 2: Computation of Effort Estimation

4.1.1 Identification of Use Case required

The objective of this step is to identify the use case that need to be implemented per iteration. For this purpose, it is necessary that the use cases have already been identified for the complete software. The identified use cases have to be included in the specifications of the software requirements. For this purpose, we used precedence diagram in which the preconditions of every use case are contained in the specifications. The use case precedence diagram is illustrated in figure 1. An arrow indicates the order in which the use cases are implemented. For instance “Use Case A” will be implemented before “Use Case 3”.

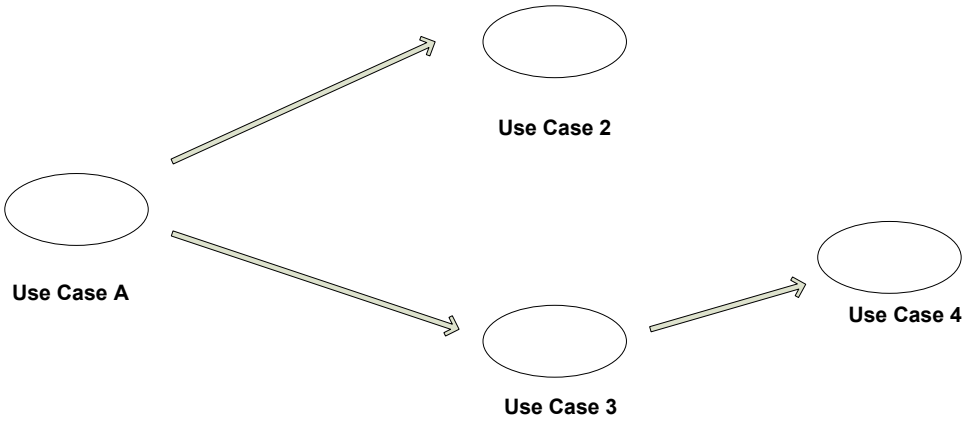

Figure 4.1: Precedence Diagram

4.1.2 Computation of Effort Estimation per Iteration

After step 1 is completed, in the step 2, Functions Points and COCOMO II are used to determine the effort that will be required for each iteration. First of all, it is necessary to determine the Unadjusted Function Points (UFP) per iteration. The reason followed to adapt this technique for iterative incremental lifecycles is to compute the resultant UFP of the complete project. The total UFP of the complete project must be equal to the sum of the Unadjusted Function Points computed separately per iteration. A challenge which must be recognized, but which this research does not address, is whether these methods will work well for the incremental code changes necessary to accommodate the additional use cases.

Total_UFP= $\sum_{i=1}^{n} \mathbf{UFP(i)}$

One of the most important contributions of this method is to determine the UFP of the Internal Logical Files (ILF) and the External Interface Files (EIF) for each use case. For this purpose, this method uses the following formula in which (File_UFP(j) = Unadjusted Function Point for a use case "j" due to files ILF/EIF, TNU(i) = total of use cases that uses a ILF/EIF "i", Weight(i)= Weight due to the complexity of ILF/EIF "i", i= ILF/ EIF used in use case "j" and j= use case involved)

$$\textbf{File\_UFP (j)} = \sum_{i=1}^{n} \frac{1}{\text{TNU(i)}} \textbf{xWeight (i)}$$

Using the results obtained from the previous formula and knowing which use case will be developed for each iteration, the UFP will be determined by means of the files present in the iteration. Next, the UFP corresponding to the transactions will be added. To accomplish this task, the following formula will be used where i=iteration, FP(i)=Total of UFP for iteration "i", File_UFP(j)= UFP for a use case "j" due to ILF/EIFs, Trans_UFP(j)= UFP due to transactions (EI,EO,EQ) and j=use case subject to be implemented in an iteration "i".

$$\boldsymbol{TUF(i)} = \sum_{j=1}^{n} [\textbf{File\_UFP(j)}] + \sum_{j=1}^{n} [\textbf{Trans\_UFP(j)}]$$

The next step uses COCOMO II and the UFP of each iteration. The effort expressed in man-month is computed per iteration and using this value, time and resources needed to complete the whole project is estimated. It is really important to point out if the context of the project is subject to change from one iteration to another (knowledge of the development platform, integration of the developer team) so that it could be useful to re-estimate the effort required by next iterations. This can be determined by reviewing files (ILF/EIF) and transactions (EI, EO, EQ) in case of change of requirements and recalculating the cost drivers suggested by COCOMO II.

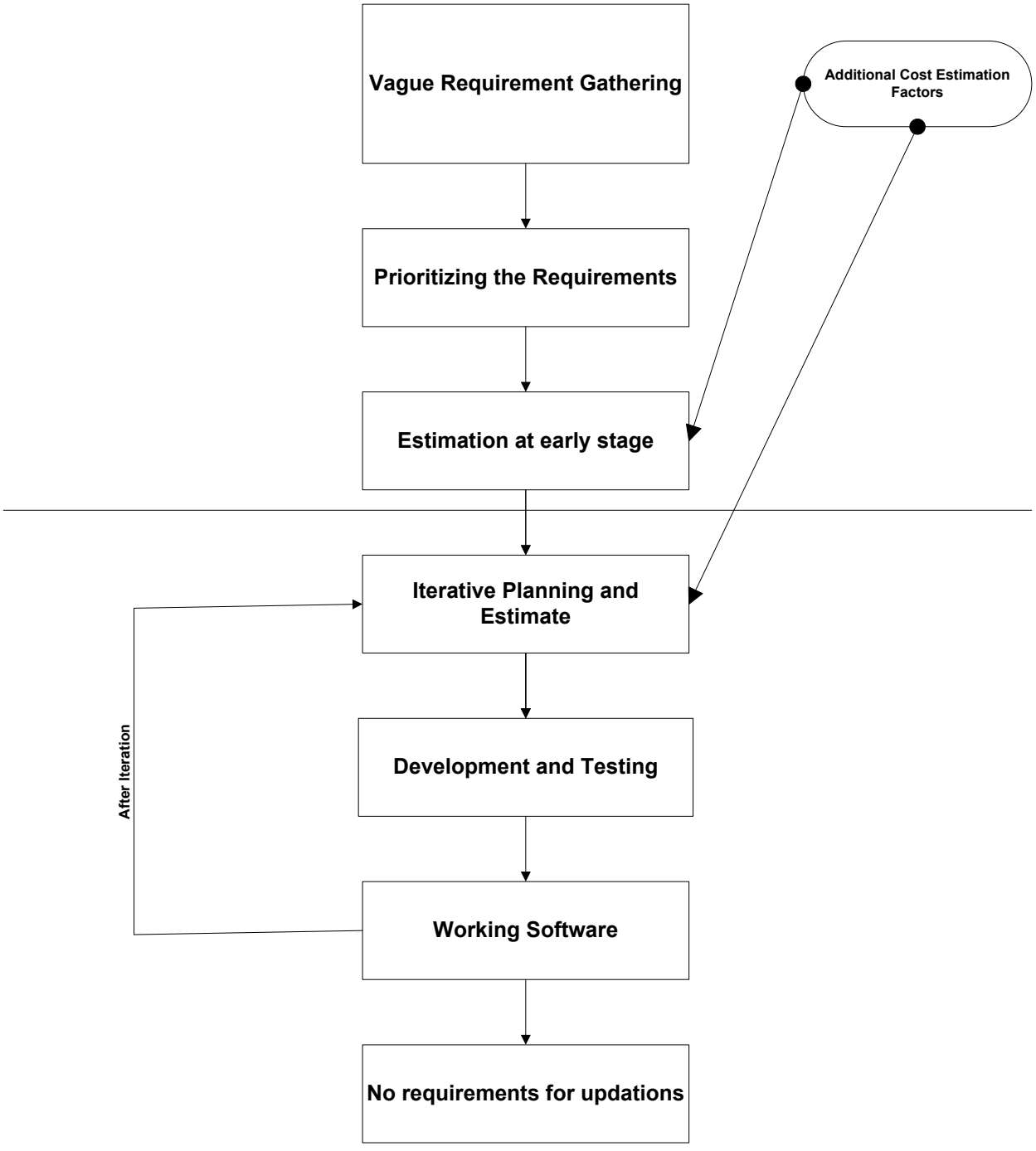

Figure 4.2: Consideration of additional cost estimation factors

4.2 Testing

For testing the proposed method for iterative software development, we used a project in the core course in the Software Engineering. In this course the student had to develop a project and the time frame of the project was 12 weeks. The students worked in projects groups consisting of five to six members developing small software systems, using an incremental development process. First of all the student did not use the whole method but only used the precedence diagram to plan the iterations of the projects. The objective was to determine the collection of certain data of the iterative software development project for the cost estimation.

4.2.1 The students and the work groups

The students are divided groups of 5 to 6. The groups were formed based upon the academic performance of the previous terms as well as the previous software development experience. Each group contains 2-3 people and they all had two years of experience in the field of software development. Table1 contains the previous knowledge and experiences that the students had prior to the beginning of the software project.

Characteristics	Experience and knowledge
Project Management	Short and medium software programming Projects
Programming Platform	C, C++, Java, C#, Python
Databases	SQL server, MS SQL server
Analysis and Design	Object oriented design
Software Estimation	No previous experience

Table 4.1: Previous knowledge and experiences

The iterative methodology is followed by each group. Before beginning the implementation phase each group had finished the Software Requirement Specification

(SRS) requirements in which they included all the possible use case scenarios and the precedence diagram for the use cases in the project.

Every group had to develop two iterations in the implementation phase. Each iteration had taken two weeks of work. The number of use cases implemented in the second and third iteration was based on the previous iterations. There were four use cases that were implemented for each iteration.

Item	Software
Documenting Software	MS office suite 2007/2010
Modeling Software	Rational Rose
Programming Language	C++, Java
Data base	Microsoft SQL server 2008
Operating System	Linux, Unix

Table 4.2: Software Used for the Projects

4.3 Experimental Results

The actual efforts are measured in men-hours. The actual effort is determine by the effective work done by each group for software design, testing and implementation.

Iteration	Group A	
	Actual Effort	UFP
1	292	187.23
2	189	195.61

Table 4.3: Actual Effort for Group A Project

Iteration	Group B	
	Actual Effort	UFP
1	322	155.13
2	277	178.30

Table 4.4: Actual Effort for Group B Project

Iteration	Group C	
	Actual Effort	UFP
1	240	175.3
2	255	138.5

Table 4.5: Actual Effort for Group C Project

Iteration	Group D	
	Actual Effort	UFP
1	295	188.4
2	169	203.5

Table 4.6: Actual Effort for Group D Project

4.3.1 Effort Estimated Per Iteration

To compute the estimated effort for iterations, the following formula used from COCOMO II:

$$\textbf{Est\_Effort (i)} = \frac{\textbf{EAF(i)}}{\textbf{i}-1} \textbf{ x } \sum_{j=1}^{i-1} \frac{\textbf{Actual\_ Effort (j)}}{\textbf{EAF (j)}}$$

Est_Effort (i) = Estimated Effort for iteration i

EAF(i) = COCOMO's EAF for the iteration i

Actual_Effort (j) = Actual effort for iteration j

The EAF factor used for calculating the estimating effort were 1.46 for first iteration and 0.8 for the second iteration.

4.3.2 Magnitude of Relative Error

A Magnitude of relative error is showing the deviation between the prediction of the formula and the observed data.

$$\textbf{Magnitude of Relative Error (MRE)} = \left| \frac{Actual\ Effort - Estimated\ Effort}{\textbf{Actual Effort}} \right|$$

Iteration	EAF Factor	Actual Effort	Estimated Effort	MRE
1	1.46	1.753	1.845	9.2%
2	0.8	1.544	1.585	2.65%

Table 4.7 Magnitude of relative error: Group A

Iteration	EAF Factor	Actual Effort	Estimated Effort	MRE
1	1.46	1.802	1.457	19.14%
2	0.8	0.998	0.875	12.3%

Table 4.8 Magnitude of relative error: Group B

Iteration	EAF Factor	Actual Effort	Estimated Effort	MRE
1	1.46	1.932	1.687	12.6%
2	0.8	1.143	1.278	11.8%

Table 4.9 Magnitude of relative error: Group C

Iteration	EAF Factor	Actual Effort	Estimated Effort	MRE
1	1.46	1.738	1.952	12.31%
2	0.8	0.893	0.972	8.8%

Table 4.10 Magnitude of relative error: Group D

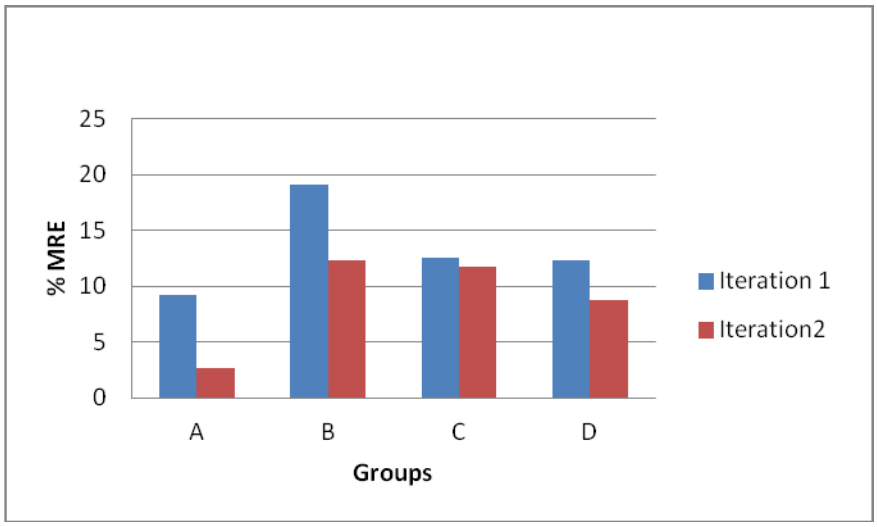

Figure 4.3: Magnitude of relative error in consecutive iteration

The recorded effort in the first iteration was useful to compute and estimate the estimation for next iteration. From the result obtained for four different groups, it is observed that in the second iteration the Magnitude of relative error is less than 13% for the groups. By computing the relative errors at each iteration, we can verify and control the certain factors by which we can improve the overall cost estimation process for the iterative software projects.

4.4 Controlled factors for the results

It is hard to generalize the result within the context of software industry. The advantage of experiments done with students is that some of the factors are controlled (knowledge, level of expertise, context of iteration) which can be difficult to control in the software industry. Some of the important factors and variables that can be controlled in student's projects are:

- Change of requirements happens throughout the project. This factor can be controlled to some extent for academic projects.

- In an industry project, the team members may be reassigned according to the expertise required for the projects. To solve the critical issue in the project, expert team member may be reassigned to the software development projects.

- Sometimes it's really hard to implement the sequence requirements in the industry projects but we can control this factor at the academic level.

4.5 Limitation of the Proposed Method

The limitations with the proposed methods are:

- Incremental delivery is a good strategy, but it does not allow for the common experience in user centered design. For the proposed method, we can easily add the use cases at each iteration but there is no way to remove the unwanted use cases at the later stage.

- Incremental delivery only assumes monolithic progress.

- The more detailed limitation of this technique is that we assume each additional use case only affects internal and external files. Actually, in many situations additional use cases can introduce additional input, output, queries or interface requirements.

CHAPTER 5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this thesis, we have proposed and developed a cost estimation tool and a method for iterative software development projects for improving the cost estimation process. The proposed cost estimation application is specifically targeted and intended for commercial software development organizations that need to estimate, analyze and utilize different vital metrics (i.e., Complexity, Size, Effort, Schedule, Manpower, Cost, etc.) involved in a software project during the discussion and planning phase. We selected cost estimation drivers (Product Attributes, Hardware Attributes, Personnel Attributes and Project Attributes) that are most likely to cause cost critical differences and also rated them for determining the impact of selected cost drivers. The importance of estimating the software metrics lies within the technique and methodology adopted. It is essential that the estimation tool be user-friendly, self-explanatory, and easy to operate. Furthermore, the results that it generates should be informative and helpful, not only in terms of project planning and management but also in providing sufficient convenience and confidence to the organization in extending near-precise commitments to their client. In other words, the cost estimation tool should leverage the organization's capabilities and strengthen it in terms of maturity, increased goodwill and credibility.

Therefore, we can hereby conclude that proposed cost estimation tool can

- Help the organizations to effectively determine complexity of a project in terms of its required team size for implementation and rigidness of requirements.

- Enable the organizations to estimate vital software metrics from different perspectives and levels of inputs, helping them to attain a wide range of input in order to give near-precise commitments to their clients.

- Enable the Project Manager(s) to calculate the size of the application in terms of lines of code through considering the functional specifications, operational constraints, organization's capability and maturity to handle that project and other significant attributes of the project.

- Strengthen the organization through estimating crucial software metrics such as Effort, Time and People required in the project, which helps the organization to confidently extend fair commitments to their clients.

- Make the organizations wise through storing the data of estimation done for previous projects, which the organization can scrutinize anytime for their information and reference purposes.

The method is proposed for the improvement in the cost estimation process for the iterative software development projects. The use case technique is implemented per iteration for the specifications of the software requirements. COCOMO II and Function Point are used to determine the effort that will be required for each iteration. The experiments were performed on student's projects and the results were analyzed. It is observed that in the second iteration the magnitude of relative error is less than 13%. By computing the relative error at each iteration, we can verify and improve the cost estimation process for the iterative software development projects. The results which we obtained cannot be generalized with the perspective of software industries because the experiments were performed on academic projects.

5.2 Future Work

This proposed cost estimation tool is built upon the essence of certain prominent software estimation and measurement techniques, wherein we have attempted to incorporate some meaningful customizations. The distinguishing features that separate this application from existing software estimation tools are its hybrid implementation methodology of estimating software metrics (such as Effort, Time and People) and its inbuilt capabilities of scheduling a project, planning human resources, and storing data for further analysis. Yet despite these accomplishments, there is still scope for further improvement in the areas of more detailed input, presentation of results, and so on, as highlighted below. We can further strengthen this application by implementing domain-specific estimation techniques. Presently, this application takes functional specifications

and operational constraints of a certain project, generically, as an input towards estimation. We can enhance this application in a way that enables the Project Manager(s) to customize the level of functional specifications and operational constraints being considered for a particular project, i.e. functionality can be added that gives Project Manager(s) the privilege to add, edit, or delete a particular input intended for the estimation.

Future version of this tool (and all other such cost estimation tools) really needs to provide not just a point estimate, but also an interval estimate. When being used in preparing a bid, for instance, without knowledge of the uncertainty in the point estimate it is hard to imagine how a project manager could be cautious and inflate the point estimate by a safety factor in order to manage the risk of unforeseen contingencies.

The proposed cost estimation method can be applied for agile software development methods because most of the projects have changing requirements which can be accommodate in the iterative software development and also required level of effort can be computed for the cost estimation whether the proposed cost estimation tool cannot be applied to agile software development.

The real accuracy of the cost estimation tool and method can be checked with respect to the software industries projects but still the project managers can help to make the proposed tool more accurate by providing their feedback to include other required inputs for increase the accuracy of proposed cost estimation tool.

Additionally, the estimated metrics can be represented graphically in such a way that their utilization is displayed in context to software project management and planning. In other words, once the metrics have been estimated, the results of their analysis in terms of project scheduling and people planning can be depicted graphically, enabling the Project Manager(s) to have a clear picture of the project implementation as a whole.

Secondly, we can build artificial intelligence into the application so that it may intelligently and effectively guide us during the estimation procedure about a particular

project through analyzing and learning from the previous data of project(s) belonging to similar domain.

Thirdly, the cost estimation for maintenance activities is still a challenge and there is a need for a cost estimation application which would help commercial software development organizations to estimate the software maintenance effort and cost for the different complexity of projects.

The other factors which should be considered are the role of prime contractors and their sub-contractors who are responsible for the range of projects in the software industries. There is still need for a process which accommodates the cost estimation process for sub-contractors in order to improve the sub-contractor cost estimation activities.

For the validation of the results of proposed method, this method needs to be applied to real- world software projects to come up with more useful results for the perspective of the commercial software development organizations. A challenge which must be recognized, which this research did not address, was whether this method will work well for incremental code changes necessary to accommodate additional use cases. For the consideration of additional use cases, Micro function point can be used for the further improvement in the method.

Finally, after reflecting the work on software cost estimation, I recommend that the following cost factors, not covered by the current models such as COCOMO I and COCOMO II, be included in cost factors for estimating the cost and effort of software project development. These cost factors are raised in this thesis for consideration for further cost estimation research but they have not been included in the proposed method.

- **Domain of the Project**

There are different activities involved for different domains of the projects which affect the cost of the projects. For example: military projects require more efforts in development and testing as compared to other commercial projects. Many military

projects involve building systems that have never been built before by anyone, so there is a challenge and risk in accomplishing the project.

- **Performance of the Project**

Performance of the project may be characterized by execution time, designing and coding standards, accuracy in outcome, etc., as per customer requirements. Thus the necessary feature must be included in the design and architecture of the software in order to achieve the required performance level. These features certainly increase the cost of the project. For example: Google search engine needs better performance for responding the queries while the performance may be less important for a commercial application. The performance of a single Google search is probably not critical. There are occasions, especially in real-time applications, where the performance of a single execution of a program is critical today (too fast may be as bad as too slow, for instance in intercept situations). Total throughput of all execution is a more common situation today and can be compensated by distributing the computation over more computers.

- **Configuration**

This is also one of the key factors in cost estimation. Configuration, in context of estimation, refers to special hardware and software requirements to run the software smoothly. For example, when users run software on a Smartphone, it is not enough that the software runs correctly without using hardware features such as audio or the multi-touch screen. Since the hardware capabilities are there, the application is not acceptable if it does not take advantage of them. The number of different platform configurations on which the software must run is also an issue. Applications for Windows and Android system must run on dozens of platforms which are unknown to the developers, whereas iPhone systems and applications only have to run on the appropriate Apple platforms. Scalability is also a configuration issue: solving very large or very small problems only, or supporting a service for a single user, through to supporting the same service for thousands of users.

- **Data Transaction**

Here, data transaction refers to the volume and frequency of data transfer from one machine to another machine. It is also access to different kinds of data. For instance, GIS data is probably not local, but access to it through mashups makes apps much more interesting (e.g. real estate or restaurant ads showing location). If the volume of transactions is high than it directly affects the effort required to develop the software and hence increase the cost.

- **Multiple Sites**

If software runs on multiple sites or many team members are to work together in distributed work environment, cost of the software increases due to the cost of communication and coordination. The more serious aspect of multiple site software is that the demands and configuration of the different sites are not the same. For instance, databases installed on each military base are not identical to each other as the data and demands of different bases are different.

- **Security concern**

Security may be considered for data security, operational security, code security, etc; depending on the stakeholder requirements. All aspects of security must be considered during the designing, coding, and implementation. They may increase the complexity in design, coding, and user interfaces and hence result in increasing the cost and duration of the project. For example, online money transactions require various levels of securities to maintain the integrity of the software.

APPENDIX A

Step 1: Start → Application's start up interface

Step 2: Select the estimation model i.e. Basic or Intermediate

If Model = Basic,
set flag → 0
else, if Model = Intermediate
set flag → 1

Step 3: Select implementation complexity

If the analyzed implementation complexity is *Simple* in terms of functional requirements & operational constraints, set class & no. of required analysts & programmers

pclass → ORGANIC
ana → 0.3
pro → 0.7

if flag = 0, set
ab → 2.4
bb → 1.05
db → 0.38

else, if flag = 1, set
ai → 3.2
bi → 1.05

If the analyzed implementation complexity is *Medium* in terms of functional requirements & operational constraints, set class & no. of required analysts & programmers

pclass → SEMI-DETACHED
ana → 0.5

pro → 0.5

if flag = 0, set
ab → 3.0
bb → 1.12
db → 0.35

else, if flag = 1, set
ai → 3.0
bi → 1.12

If the analyzed implementation complexity is *High* in terms of functional requirements & operational constraints, set class & no. of required analysts & programmers

pclass → EMBEDDED
ana → 0.7
pro → 0.3

if flag = 0, set
ab → 3.6
bb → 1.20
db → 0.32

else, if flag = 1, set
ai → 2.8
bi → 1.20

Step 4: Calculate the implementation's size through Function Point

4.1: Specify functional characteristics

Rate the no. of *User Inputs* that are involved in the implementation according to the selected complexity.

if rating = Simple, set var1 → 2
else, if rating = Average, set var1 → 4
else, if rating = Complex, set var1 → 6

Rate the no. of *User Outputs* that are involved in the implementation according to the selected complexity.

if rating = Simple, set var2 → 3
else, if rating = Average, set var2 → 5
else, if rating = Complex, set var2 → 7

Rate the no. of *User Queries* or reports that are involved in the implementation according to the selected complexity.

if rating = Simple, set var3 → 2
else, if rating = Average, set var3 → 4
else, if rating = Complex, set var3 → 6

Rate the no. of *Files or Databases* that are involved in the implementation according to the selected complexity.

if rating = Simple, set var4 → 5
else, if rating = Average, set var4 → 10
else, if rating = Complex, set var4 → 15

Rate the no. of *External Interfaces* that are involved in the implementation according to the selected complexity.

if rating = Simple, set var5 → 4
else, if rating = Average, set var5 → 7
else, if rating = Complex, set var5 → 10

4.2: Specify more functional & operational constraints

Rate the *Reliable Backup & Recovery* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var6 → 1

else, if rating = Low, set var6 → 2

else, if rating = Medium set var6 → 3

else, if rating = High, set var6 → 4

else, if rating = Very High, set var6 → 5

Rate the *Required Data Communications* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var7 → 2

else, if rating = Low, set var7 → 3

else, if rating = Medium set var7 → 4

else, if rating = High, set var7 → 5

else, if rating = Very High, set var7 →6

Rate the *Distributed Processing* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var8 → 1

else, if rating = Low, set var8 → 2

else, if rating = Medium set var8 → 3

else, if rating = High, set var8 → 5

else, if rating = Very High, set var8 → 7

Rate the *Critical Performance* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var9 → 3

else, if rating = Low, set var9 → 4

else, if rating = Medium set var9 → 5

else, if rating = High, set var9 → 6
else, if rating = Very High, set var9 → 7

Rate the *Operational Environment Complexity* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var10 → 2
else, if rating = Low, set var10 → 3
else, if rating = Medium set var10 → 4
else, if rating = High, set var10 → 5
else, if rating = Very High, set var10 → 6

Rate the *Online Data Entry* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var11 → 3
else, if rating = Low, set var11 → 4
else, if rating = Medium set var11 → 5
else, if rating = High, set var11 → 6
else, if rating = Very High, set var11 → 7

Rate the *Multiple Screens & Concurrent* Operations constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var12 → 1
else, if rating = Low, set var12 → 2
else, if rating = Medium set var12 → 7
else, if rating = High, set var12 → 8
else, if rating = Very High, set var12 → 9

Rate the *Online Updating of Master File* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var13 → 2

else, if rating = Low, set var13 → 4

else, if rating = Medium set var13 → 6

else, if rating = High, set var13 → 8

else, if rating = Very High, set var13 → 10

Rate the *Inputs, Outputs & Database Complexity* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var14 → 3

else, if rating = Low, set var14 → 5

else, if rating = Medium set var14 → 7

else, if rating = High, set var14 → 9

else, if rating = Very High, set var14 → 11

Rate the *Internal Processing Complexity* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var15 → 3

else, if rating = Low, set var15 → 7

else, if rating = Medium set var15 → 9

else, if rating = High, set var15 → 10

else, if rating = Very High, set var15 → 11

Rate the *Code Reusability* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var16 → 1

else, if rating = Low, set var16 → 2

else, if rating = Medium set var16 → 3

else, if rating = High, set var16 → 4

else, if rating = Very High, set var16 → 5

Rate the *Installation & Configuration* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var17 → 3
else, if rating = Low, set var17 → 7
else, if rating = Medium set var17 → 11
else, if rating = High, set var17 → 12
else, if rating = Very High, set var17 → 15

Rate the *Concurrent Pipelined Execution* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var18 → 1
else, if rating = Low, set var18 → 5
else, if rating = Medium set var18 → 6
else, if rating = High, set var18 → 7
else, if rating = Very High, set var18 → 9

Rate the *Ease of Use & Modification* constraint according to its importance and complexity & functional characteristics of the proposed implementation.

if rating = Very Low, set var19 → 6
else, if rating = Low, set var19 → 7
else, if rating = Medium set var19 → 9
else, if rating = High, set var19 → 10
else, if rating = Very High, set var19 → 11

4.3: Specify development technology

if technology = C, set var20 → 1
if technology = C++, set var20 → 2
if technology = Java, set var20 → 3

Step 5: Calculate the value of Function Point

Set fp → $(0.01 * \sum var_i (i = 6 \text{ to } 19) + 0.65) * \sum var_i (i = 1 \text{ to } 5)$

Calculate and determine the size of application of solution-implementation of size in the estimated lines of code (LOC)

Set ELOC → FP* var20

if ELOC <= 8, set size → Small

else, if 8 < ELOC <=32, set size → Intermediate

else, if 32 < ELOC<=128, set size → Medium

else, if ELOC > 128, set size → Large

Step 6: Specify various implementation attributes and cost drivers involving other different functional & operational characteristics in order to calculate the Effort Adjustment Factor.

6.1: Specify Product Attributes

Rate *Required Software Reliability* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp1 → 0.75

else, if rating = Low, set temp1 → 0.88

else, if rating = Nominal, set temp1 → 1.00

else, if rating = High, set temp1 → 1.15

else, if rating = Very High, set temp1 → 1.40

else, if rating = Extra High, set temp1 → 1.60

Rate *Application Database Size* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp2 → 0.77

else, if rating = Low, set temp2 → 0.94

else, if rating = Nominal, set temp2 → 1.00

else, if rating = High, set temp2 → 1.08

else, if rating = Very High, set temp2 → 1.16

else, if rating = Extra High, set temp2 → 1.28

Rate *Product Complexity* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp3 → 0.70

else, if rating = Low, set temp3 → 0.85

else, if rating = Nominal, set temp3 → 1.00

else, if rating = High, set temp3 → 1.15

else, if rating = Very High, set temp3 → 1.30

else, if rating = Extra High, set temp3 → 1.65

6.2: Specify Hardware Attributes

Rate *Runtime Performance* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp4 → 0.76

else, if rating = Low, set temp4 → 0.88

else, if rating = Nominal, set temp4 → 1.00

else, if rating = High, set temp4 → 1.11

else, if rating = Very High, set temp4 → 1.30

else, if rating = Extra High, set temp4 → 1.66

Rate *Memory Constraints* attribute according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp5 → 0.75

else, if rating = Low, set temp5 → 0.85

else, if rating = Nominal, set temp5 → 1.00

else, if rating = High, set temp5 → 1.06

else, if rating = Very High, set temp5 → 1.21

else, if rating = Extra High, set temp5 → 1.56

Rate the *Virtual Machine Environment Volatility* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp6 → 0.80
else, if rating = Low, set temp6 → 0.87
else, if rating = Nominal, set temp6 → 1.00
else, if rating = High, set temp6 → 1.15
else, if rating = Very High, set temp6 → 1.30
else, if rating = Extra High, set temp6 → 1.66

Rate *Required Turnaround Time* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp7 → 0.71
else, if rating = Low, set temp7 → 0.87
else, if rating = Nominal, set temp7 → 1.00
else, if rating = High, set temp7 → 1.07
else, if rating = Very High, set temp7 → 1.15
else, if rating = Extra High, set temp7 → 1.60

6.3: Specify Personnel Attributes

Rate *Analyst Capability* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp8 → 1.46
else, if rating = Low, set temp8 → 1.19
else, if rating = Nominal, set temp8 → 1.00
else, if rating = High, set temp8 → 0.86
else, if rating = Very High, set temp8 → 0.71
else, if rating = Extra High, set temp8 → 1.90

Rate *Application's Experience* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp9 → 1.29
else, if rating = Low, set temp9 → 1.13
else, if rating = Nominal, set temp9 → 1.00
else, if rating = High, set temp9 → 0.91
else, if rating = Very High, set temp9 → 0.82
else, if rating = Extra High, set temp9 → 1.40

Rate *Software Engineer Capability* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp10 → 1.42
else, if rating = Low, set temp10 → 1.17
else, if rating = Nominal, set temp10 → 1.00
else, if rating = High, set temp10 → 0.86
else, if rating = Very High, set temp10 → 0.70
else, if rating = Extra High, set temp10 → 0.75

Rate the *Virtual Machine Experience* attribute according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp11 → 1.21
else, if rating = Low, set temp11 → 1.10
else, if rating = Nominal, set temp11 → 1.00
else, if rating = High, set temp11 → 0.90
else, if rating = Very High, set temp11 → 0.75
else, if rating = Extra High, set temp11 → 1.70

Rate *Programming Language Experience* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp12 → 1.14
else, if rating = Low, set temp12 → 1.07
else, if rating = Nominal, set temp12 → 1.00

else, if rating = High, set temp12 → 0.95

else, if rating = Very High, set temp12 → 0.66

else, if rating = Extra High, set temp12 → 0.70

6.4: Specify Project Attributes

Rate *Software Tools Usage* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp13 → 1.24

else, if rating = Low, set temp13 → 1.10

else, if rating = Nominal, set temp13 → 1.00

else, if rating = High, set temp13 → 0.91

else, if rating = Very High, set temp13 → 0.82

else, if rating = Extra High, set temp13 → 0.90

Rate *Application of Software Engineering Methods* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp14 → 1.24

else, if rating = Low, set temp14 → 1.10

else, if rating = Nominal, set temp14 → 1.00

else, if rating = High, set temp14 → 0.91

else, if rating = Very High, set temp14 → 0.83

else, if rating = Extra High, set temp14 → 0.90

Rate *Development Schedule* attributes according to its importance based upon the complexity & size of the prospective solution implementation

if rating = Very Low, set temp15 → 1.23

else, if rating = Low, set temp15 → 1.08

else, if rating = Nominal, set temp15 → 1.00

else, if rating = High, set temp15 → 1.15

else, if rating = Very High, set temp15 → 1.40

else, if rating = Extra High, set temp15 → 1.60

Set EAF→ temp1 * temp2 * temp3 * temp4 * temp5 * temp6 * temp7 * temp8 * temp9 * temp10 * temp11 * temp12 * temp13 * temp14 * temp15

EAF=Effort Adjustment Factor

Step 7: Calculate estimated Effort, Time & People required

If flag = 0

Set effort → $eloc^{bi}$ * ab

else, if flag = 1

Set effort → ai * $eloc^{bb}$ * EAF

Set time → $effort^{db}$ * cb

Set people → effort / time

BIBLIOGRAPHY

[1] Software Development Cost Estimation. Phillippe Kruchten, University of British Columbia

[2] A. Issa, Algorithmic software cost estimation model for early stage of software development, international journal of academic research, March 2011

[3] Linda M. Laird, "The Limitations of Estimation," IT Professional, vol. 8, no. 6, pp. 40-45, Nov./Dec. 2006, doi:10.1109/MITP.2006.149

[4] V.Khatibi and D.Jawawi, Software Cost Estimation Method: A Review. *Journal of emerging trends in computing and information sciences*, 2011

[5] K. Kavoussanakis and T. Sloan. UKHEC Report on Software Estimation, Dec 2001. accessed on http://www.ukhec.ac.uk/publications/reports/estimation.pdf

[6] Barry Boehm, "*Safe and Simple Software Cost Analysis,"* IEEE Software, vol. 17, no. 5, pp. 14-17, September/October, 2000

[7] K. Moloekken-OEstvold, M. Joergensen, S.S. Tanilkan, H. Gallis, Lien, A.C. Lien, S.W. Hove, A survey on software estimation in the Norwegian industry, *Proceedings 10th International Symposium,* pp. 208- 219, 14-16. Sept. 2004

[8] M. Nasir. A Survey of Software Estimation Techniques and Project Planning Practices. *Proceedings of the Seventh ACIS International Conference on Software Engineering.* pp. 305-310, IEEE Computer Society. 2006

[9] Jenkins, A.M., J.D. Naumann, and J.C. Wetherbe, *Empirical Investigation of Systems Development Practices and Results.* Information & Management, 1984. **7**: p. 73- 82.

[10] Phan, D., Information Systems Project Management: an Integrated Resource Planning Perspective Model, in *Department of Management and Information Systems*. 1990, Arizona: Tucson.

[11] F. Bergeron and J.-Y. St-Arnaud, Estimation of Information Systems Development Efforts. *A Pilot Study. Information & Management,* page 239-254, 1992

[12] Heemstra, F.J. and R.J. Kusters. Controlling Software Development Costs: A Field Study. In *International Conference on Organisation and Information Systems*. 1989. Bled, Yugoslavia.

[13] A.L. Lederer, and J. Prasad, Causes of Inaccurate Software Development Cost Estimates. *Journal of Systems and Software*, pp. 125-134, 1995

[14] A.L. Lederer, and J. Prasad, Information systems software cost estimating: a current assessment. *Journal of Information Technology*, pp. 22-33, 1993

[15] Sauer, C. and C. Cuthbertson, *The State of IT Project Management in the UK 2002-2003*. Templeton College, University of Oxford.

[16] G.N. Parkinson, *Parkinson's Law and Other Studies in Administration*, Houghton-Miffin, Boston, 1957

[17] Roger S. Pressman**, "***Software Engineering, A Practitioner's Approach***"** Sixth Edition, McGraw-Hill, NY, 2005.

[18] C. Ravindranath Pandian, *Software Metrics A Guide to planning, Analysis and Application,* India, 2004

[19] Steve McConnell. *Software Project Survival Guide*. Microsoft Press, 1998.

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

[21] I. Sommerville. *Software Engineering, Sixth Edition*. Addison-Wesley Publishers Limited, 2001.

[22] W.S.Humphrey. Your Date or Mine. In *The Watts New Collection*. Software Engineering Institute, Carnegie Mellon University, http://interactive.sei.cmu.edu/, 2001.

[23] Steve McConnell. *Rapid development: taming wild software schedules*. Microsoft Press, 1996.

[24] L. Putnam and W Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.

[25] C. Jones. *Applied Software Measurement*. McGraw Hill, 1997.

[26] R Park. The Central Equations of the PRICE Software Cost Model. In *4th COCOMO Users' Group Meeting, November 1988*.

[27] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[28] Zia, Z.; Rashid, A.; uz Zaman, K.;, "Software cost estimation for component based fourth-generation-language software applications," Software, IET , vol.5, no.1, pp.103-110, February 2011 doi: 10.1049/iet-sen.2010.0027

[29] H. Leung and Z. Fan, Software Cost Estimation. Handbook of software engineering and knowledge engineering, world scientific publications company, River Edge, NJ, 2002

[30] L.H. Putnam and W. Myers, "Manager: How Solved is the Cost-Estimation Problem?, published at IEEE Software, 1997, pp.105-108.

[31] Barry Boehm, "Safe and Simple Software Cost Analysis," IEEE Software, vol. 17, no. 5, pp. 14-17, Sep./Oct. 2000, doi:10.1109/52.877854

[32] http://www.ere.net/2003/07/08/characteristics-of-a-good-metric. Last accessed: May 20, 2011

[33] Mauricio J. Ordonez, Hisham M. Haddad, "The State of Metrics in Software Industry," itng, *Fifth International Conference on Information Technology*: *New Generations*, pp.453-458, 2008

[34] Evolutionary Software Development, The Research and technical organization (RTO) of NATO, published in August 2008

[35] Barry W. Boehm, C.M. Abts and E.K. Bailey, "COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings, Proceedings ESCOM-SCOPE, pp. 325-333, 2000

[36] The International Function Point User Group (IFPUG), Function Point Counting Practices Manual-Releases 4.1, USA, 1999

[37] J. martin, *"Software Maintenance: The Problem and Its Solution"*. Prentice Hall, 1983, pp. 472

[38] http://www.softstarsystems.com/overview.htm. Last accessed: August, 13, 2011

[39] C. Jones, Strengths and Weaknesses of Software Metrics, Software Productivity Research, Version 5, March 22, 2006