Fashina Alfred

CHALLENGES IN SOFTWARE PROJECT COST ESTIMATION: A COMPARATIVE CASE STUDY

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

ABSTRACT

Fashina, Alfred
Challenges in the process of Estimating Software Cost
Jyvaskyla: University of Jyvaskyla, 2021, 50 p.
Information System Science, master's thesis
Supervisor: Pekka, Abrahamsson

Estimating the cost, effort, and size to complete a software project is one of the most difficult and confusing tasks confronted by software project managers. Though, an early estimate is very crucial when bidding for contracts or determining whether the project viable, it's accuracy cannot be guaranteed because of factors like incomplete requirements, inadequate information from past projects and the experience of the estimator.

Accurate software cost estimate can help the developer make more logical decisions in planning, scheduling, allocating resource, and monitoring the project progress. Considering all the estimation models developed by various researchers, it is inevitable to say that there has not been a perfect estimation method that solves all estimation problem.

The first part of this thesis provides a general overview of software estimation and some models, which are classified as algorithm and non-algorithm models. The second part is a comparative case study research, which emphasizes on two non-algorithm model, *Top-down* and *Bottom-Up method* in comparison with the estimate gotten from a software development project.

The main result of this study is that it is almost impossible to evaluate an accurate and error-free estimate at the beginning of a software project. Combining two or more estimation models at the beginning of the project and enhancing the estimate as the project progresses could give the better estimate, but other factors like risk assessment, resetting expectation, unexpected unknowns and exploring the use of automation should also be considered.

Keywords: Software cost estimation, cost overrun, software project, size estimation, Algorithm and Non-Algorithm methods

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

TABLE OF CONTENTS

1 INTRODUCTION

Human's dependency on computer has increased greatly, so much that it has become part of our everyday life. This has also increased the need for faster functionality, smaller interface and secured platforms. Software companies are aiming to meet this need while also minimising development cost and delivery time: (Stutzke, 1996). To achieve this, it is important accurately estimate the effort required to complete the software project and meet the expected completion date.

Software estimation has been an essential and difficult procedure since the beginning of the computer era. The bulk of the cost of software development is calculated as human effort in relation to time (usually in persons-months). Effective software cost estimates are critical to survival of most organizations because it helps to determine what resources to commit to the project, how well to use them, and what to prioritize. It is also be used for generating request for proposals, contract negotiations, scheduling, monitoring and control: (Zia, Rashid & Zaman, 2011)

Often, most unfinished software projects have been blamed for inadequate requirements, experience of developers and estimator and cost overrun: (Hihn & Habibagahi, 2000). Software estimates made in the early stages of a product development are usually wrong because of many elements of uncertainty, which often lead to over or under-estimation of software size and effort: (Kruchten, 2007)

Research on software cost estimation started with software companies and military organizations that develop large software systems: (Jones, 2005). These estimates are used to define budgets, schedules, risks, and resource allocation: (Boehm, Abts & Chulani, 1998). Most of the commonly used estimation models are either algorithmic or non-algorithmic, but new models that use machine learning approaches are being researched: (Stamelos, Angelis, Morisio, Sakellaris & Bleris, 2003).

A good software cost estimate should have the following attributes: (Royce, 1998)

- It is accepted by all stakeholders as realizable.
- It is based on a well-defined software cost model with a credible basis.
- It is based on a database of relevant project experience.

- It is defined in enough detail so that its key risk areas are understood, and the probability of success is objectively attainable.

1.1 Research Problems and questions

The main research question of this thesis is to analyse the challenges encountered in the process of estimating a software development project, comparing two of the non-algorithm models with the real-world data. To support the answers to the question above, the following sub-questions are formulated:

- What is the best estimation method for any software project?
- What are the key reasons for cost overrun in developing large software?
- Is it possible to estimate a software project, by using the Bottom-Up or the Top-down method alone?

This research attempts to provide answers to these questions by

- reviewing literatures in the field of software cost estimation and
- comparing the Bottom-Up and the Top-Down method with research analysed.

1.2 Structure of the thesis

Chapter 1 introduces the background of the study, the research problems, key objectives, the motivation, scope, and the structure of the study.

Literature review is conducted in Chapter 2. This chapter introduces fundamental concepts in software cost estimation, classification of software metrics, challenges encountered during the process of estimating software cost.

Chapter 3 introduces the different kinds of software cost estimation techniques.

Chapter 4 introduces the overview of the research methodology applied in the empirical part of the study. It explains the choice of the research approach and design. It also discusses the limitations and reliability of the research method.

Chapter 5 describes the data to be used for the case study analysis.

Chapter 6 presents the use cases and empirical analysis of the research data described in chapter 5.

Chapter 7 discusses the primary empirical contribution of the analysis in chapter 6, and its implication to the research.

Chapter 8 concludes the research. The answer to research questions are presented, limitation of the study and further research opportunities on subject matter were discussed.

2 LITERATURE REVIEW

Despite all the software cost estimation methods developed, there is still no straightforward way to generate an accurate estimate of the effort, time or cost required to complete a software project (Bill, 2020). One research report outlined that barely 5% of software projects are completed on time and within budget. Another indicates that less than 1% of commercial software projects are completed on time, within budget and according to specifications. In addition to that, just about 3 out of 4 software projects begun are either never completed or cost more than estimated. (Zawrotny, 1995). This was supported by McConnell (1998), who reported that more than half of software projects either overrun their budget, get cancelled or delivered late.

According to Steve McConnell (2006), a good estimate is an essential part of project management which provides a clear view of the project structure, thereby giving managers the resources to make decisions and have the desired result. Though, it is difficult to generate a detailed estimate until each feature is understood, he suggested that an estimate with 75% accuracy is sufficient to start a project.

These studies shows that it is almost impossible to estimate software development costs accurately at the beginning of a project. This also indicates that over-estimating or under-estimating of a project are common occurrence that happen in software development. For example, an underestimated project could lead to under staffing, make developers work harder than required, reduce the time that could be assigned for testing and creativity, and bad quality. On the other side, overestimation could stretch a project to take at least as long as it was estimated for, even when it can be completed earlier and over budgeting. (Linda, 2006)

Several reasons were proposed by different literatures on why many projects overrun its estimate. The factors as listed by Linda (2006) include the lack of training and experience of developers and estimators, indecision of the acceptable deliverables, and changing of the requirements. The other reasons identified by Linda are difficulty managing the schedule of the project as the requirements change, unreliable expectation, and insufficient resources for the project.

Khatibi and Jawawi (2011) conducted an intensive research, using 2100 internet websites and came up with several reasons for software projects failure. The most popular reasons found are insufficient or defective requirements, poor planning, and inaccurate estimation. Boehm (1984) suggested that lack of clear understanding of the software requirement and misjudging the size and required effort for the software projects are the main reasons for inaccurate estimations.

In this study, software project estimation can be regarded as one of the following.

- effort hours estimation
- project duration estimation
- software cost estimation

Some authors suggested that the main problem with software project estimation is the lack of distinct regulation and standards to adhere to during the overall process of software development. This might create a guide to detect and resolve the inaccuracy in an estimate is to recognize the three related quantities, i.e., functional specification, cost, and delivery time.

2.1 Size Estimation

One of the main reasons why software projects fail is the inability to accurately determine the size of the project. According to Campbell (1995), poor size estimates are usually main cause of cost and schedule overruns. To resolve the issue of accurately calculating the size of a software project, it is recommended to use a variety of software sizing techniques. Depending on a single technique has been noticed to be a major reason for cost overrun and late delivery. (Watt, 1989).

Most complex and large software projects have been underestimated, because it is demanding to accurately estimate the actual size. (Stutzke, 2005). Many large projects are regarded as high risk because a change in the requirement could be difficult and expensive. Some large software project failure could lead to billion of dollars in loss. (Charette, 2005). It might also require authorization from many stakeholders before such changes can be accepted. There is also the possibility of project failure due to changing user expectations and requirements, friction caused by undefined roles among developers and so many unforeseen events.

There are two types of measurements for software product size. These include *Line of Code* and *Function Point.* However, there are other not too common ones which include *Object Points, Application Points, Predictive Object Points* and *Unified Modelling Languages.*

2.1.1 Lines of Code (LOC)

The Lines of Code (LOC) is the number of source statements delivered at the completion of the software project. It is one of the most widely used measurement for software size and complexity: (Rosalind, Pfleeger & Wu, 2005). One problem with using Lines of Code (LOC) as a metric of measurement for software size is that it cannot be used to estimate projects with multiple programming language since each language has its own pattern and syntax. Other issues with LOC are that it does not take efficiency, accuracy, usability, execution speed and quality of the code into consideration: (Stevenson, 1995).

The two types of LOC measures are the *physical* and *logical* LOC. The physical LOC is an easy way of counting the lines of code. It is counting all the lines of the program's source code including comments and blank lines. On the other hand, the logical line of code is more practical than the physical line of code. It is regarded as all executable lines or statement created that performs a function: (Nguyen, Deeds-Rubin, Tan & Boehm, 2007)

Although many literatures have been written that uses LOC as the size measure, it is difficult to count the lines of code in the development process and there isn't an accepted counting standard: (Touesnard, 2004).

2.1.2 Function Points

Function Points is a measure of the amount of functionality delivered by the software in a project. According to Allan Albrecht (1979), Function Point is categorized into: *Outputs, Inquiries, Inputs, Internal files, and External files (or interface)*. Function Points is useful because it can be obtained from detailed requirements. However, it cannot be used for assessing the size of embedded system.

Although function points support software size estimates, it is still difficult to estimate at the beginning of the project and can be cumbersome when assessing an embedded system: (Symons, 1988). Though, difficult to estimate at the beginning of a software development process, but it remains valuable as the requirements becomes explicit. Like LOC, function points are also affected by changing requirements: (Garmus & Herrod, 2001)

Both sizing methods have their advantages and disadvantages, which cannot be ignored and could be used to complement each other. These sizing methods are dependent on the knowledge of the system, experience of the developer writing the code, and system composition in general: (Symons, 1991)

2.2 Challenges in the process of estimating Software Cost

The challenges in accurately estimating software size, time or effort certainly affect the cost of the software. There are various challenges in estimation, each of which is related to uncertainty and occur at several places throughout a project's life cycle. Every time a decision is made concerning the software project, an element of complication or difficulty is introduced into the estimation process: (Eberendu, 2014).

The most difficult aspect of estimation occurs when cost estimates must be made at the beginning of the software project. For most new project, an estimate is needed at the early stage of the project, to have an idea of how much will be needed to complete the project.

For projects that have already started, changes to the requirements, affects the estimate greatly and could present a bigger problem to its completion if it is not managed early. The following are some of the challenges encountered in the process of estimating software project.

2.2.1 Incomplete Requirements

Incomplete or inadequate requirements is regarded as the major reason why cost estimates are inaccurate. This problem could be regarded as the most difficult to ignore because most users do not really understand their requirements during the early stages of the project. Software projects are often undertaken when there is a recognition of need, while the requirements specification at a sufficiently detailed level unavailable: (Strike & Emam, 2001). Estimates made at this stage have a high likelihood of error. A fact that must be accepted is that a complete statement of the requirements cannot be defined before development begins: (Humphrey, 1989).

For identical projects, even when the software system being developed is almost identical to a previously developed system, the requirements or features will be different because no two software projects are the same: (Hull, 2009). As a project evolves, product owner gains a clearer and better understanding of the problem and can create detailed requirements. The inadequacy or experience of the writer of the requirement could also affected the cost of the software. Many written requirements are either bias, obsolete, or inconsistent because the writer is unable or unwilling to use the latest technology in achieving their goals or just don't have the required skills and experience: (Boehm, 2010).

2.2.2 Maintenance of developed software:

Software maintenance cost is often ignored during the estimation and can be significantly higher than development costs if it is not managed properly. Ironically, maintenance costs are much easier to estimate than the overall cost of developing software but are often neglected: (Albert, Lederer & Jayesh, 1992).

Though estimating the maintenance cost may be an easier task, but there is the tendency that a maintenance team can inherit an incomplete or unmaintainable software from the development organization: (Koskinen, 2010) Additionally, it is difficult to predict if the development team has designed the system to be maintainable. Though design documentation might have been provided, there is no assurance that it is detailed enough, especially in the situation where they are been pressurized to complete the project as soon as they can: (Dehaghani & Hajrahim, 2013)

The problems stated above are more evident in projects that have a separate development and maintenance team. For example, a development team project manager's responsibility end when the completed system is delivered within the specified budget and time, therefore having no stake in the maintenance effort: (Nguyen, 2010).

2.2.3 The Project Procurement Procedure

Procurement, which is usually conducted at the early stage of the software project, can be challenging for both the procuring team, and the developer. At the beginning of the procurement process, bids are received, and a suitable developer team is chosen to complete the project with the accepted estimate.

Some procurement team have a two-stage estimation process: the pre- and post-contract estimates. The pre-contract estimate is used for bidding for the contract. This strategy is generally called "bid to win" approach. Such bids are often prepared quickly from requirements which were often vague with no technical details. Sometimes, the procurement team is forced estimate as low as possible for various tasks by the management.

Once a company is awarded the contract, it frequently performs another more detailed estimate which is considered the post estimate or the real, which is regarded as realistic. If the "real" estimate is higher than the "bid to win" estimate, it might become an issue that could be difficult to resolve: (Novack, 1991)

Some procurement team might suggest adding enhancements or finding problems with the requirements while others might reduce the functionality of the system to balance the budget. Some small companies might just accept the project as a loss and hope to use the project to build their portfolio: (Hung, 2006)

2.2.4 Tracking the Progress of the Project

Software costs cannot be controlled unless the software costs and progress are measured. Most software task are considered complete when the person responsible for the task or the head of the development team, declare it to be complete.

Milestone and technical reviews are the typical techniques used by procurement team to gain control over the development process. Though, milestone reviews are necessary but are not by themselves sufficient to monitor progress on a project: (Boehm, 2010).

2.2.5 Lack of Historical Data

Organization involved in the development of a new software needs information about previous projects to estimate accurately what will happen in its next development project. This information or data cannot be solely relied on for estimation because no two software are the same: (Charette, 2005). For small projects, relying on historical data and the experience of key people in the organization could still provide an accurate estimate but almost impossible for larger project that are more complex, and the knowledge is distributed among larger numbers of people: (McConnell, 1998).

3 SOFTWARE COST ESTIMATION TECHNIQUES

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, various models have been significantly developed and used for estimating cost. These cost estimation methods are classified under two branches: *Algorithmic* and *Non-Algorithmic.* The Algorithmic methods are based on simple arithmetic formulae using summary statistic. (Donelson, W. 1976), while the Non-Algorithmic method rely on data from previous software projects to develop the estimate.

3.1 Non-Algorithmic Methods

The non-algorithmic methods involve using previous similar software projects and experience from such project to derive the estimation. In this method, estimation is only completed based on analysis of previous software projects. Some non-algorithmic methods are described below:

3.1.1 Analogy estimation methods

This method involves comparing by analogy with a completed project to compare their actual costs to an estimate of the cost of a similar new project. (Shepperd & Schofield, 1997). Generally, since there are rarely two perfectly matched projects, some adjustment is needed to fit both projects together. The drawback of this method is that the estimate gotten will be subjective and challenging because two projects that look similar are always different. Estimating by analogy can be straightforward but it is not as easy as it looks.

Some advantages of this method are:

- The estimation is based on actual project characteristic data.
- The estimator's experience can be used to improve the estimate.
- For a fairly small project, the distinction between the completed and the proposed project can be identified, and difference reconciled.

Some disadvantages of this method,

- The choice of variables is restricted to information and data from the previously completed project and any adjustment could alter the similarities between both projects.
- This method cannot be use for every project.
- This method limit creativity.

3.1.2 Expert Judgment

This method involves consulting one or more experts to derive an estimate. This method can be relatively accurate if the estimator has significant knowledge about both the project domain, and the estimation process: (Hihn & Habib-agahi, 1990). Sometimes, expert judgment could be an educated guess supported by a variety of tools to predict the amount of effort or cost required to complete the project: (Kruchten, 2007). For example, an expert might access the database of past projects to understand the new project and use the experience of the system domain to develop an estimate.

Some advantages of this method are:

- The experts can manage the differences between past project experience and requirements of the proposed project to create a better estimate.
- Using expert judgement method can help leverage new technologies, architectures, applications, and languages.

The disadvantages include:

- It is difficult for the expert to quantify human efficiency of the developers.
- Expert may be some biased towards a certain way of estimating and that could be detrimental to an organization that doesn't work that way.

3.1.3 Top-down Method

In the Top-down approach, the total cost estimate for the project is derived at the early stage of the project. This approach starts at the system level, by examining the overall functionality of the product and later broken down to the various sub-components of the system: (Liming, 1997).

Top-down estimating method is also called Macro Mode. It is more applicable to early cost estimation when only general properties are known. This method is very useful because it is a quick way to have a rough idea of how much the total project might cost: (Iqbal, Idrees, Sana & Khan, 2017).

Some advantages of this method are:

- It focuses on system-level activities such as integration, documentation, configuration management, etc.
- It requires minimal project detail.
- It is faster to develop and easier to implement.

The disadvantages are:

- It does not recognize smaller and technical details of the software that might escalate budget and lead to project failure.
- It cannot be used for large software projects.

3.1.4 Bottom-up Method

The Bottom-Up method is the opposite of the Top-Down method. It starts at the small component level and the results added together to produce an estimate for the overall project: (Leung & Zhang, 2001)

Some advantages are:

- It helps developers have a feel of the overall structure of the project even before the start of the project.
- It is more stable because the project flaws in the various components can be detected early.

The disadvantages:

- It could still be incorrect because the detailed requirements are usually unknown at the early stage of the project.
- It is time-consuming to develop.
- It is not possible to estimate unknown or unexpected problems.

Other non-algorithm methods, like price-to-win, Parkinson methods, Nelson model can also be used to estimate the cost of software. In practice, two or more methods are used together to derive the best estimate for the project: (Casper, 2007)

3.2 Algorithmic Methods

Algorithm model uses some derived mathematical equations to predict project cost, based research and historical data using metrics such as Lines of code (LOC) and number of functions. Many algorithmic methods studied and developed includes, the COCOMO model, Putnam model, and function points-based models: (Khatibi & Jawaw, 2011).

3.2.1 Constructive Cost Model (COCOMO) Method

It was first published in Boehm's 1981 book "*Software Engineering Economics*" as a model for estimating effort, cost, and schedule for software projects. It is the most-used software cost and schedule estimation model (Boehm, B.W 1995). The model uses basic equation with parameters that are derived from historical project data and current project. In 1995, *COCOMO II* was developed and finally published in 2000 in the book *Software Cost Estimation with COCOMO II*.

$E = a(KLOC)^b$ MM
Time (D) = $c(E)^d$ Month(M)
Person required = E/D

- E = Total effort required for the project in man-Months (MM)
- D = Total time required for project development in Months (M)
- KLOC = the size of the code for the project in kilo lines of code
- a, b, c, d = The constant parameters for software project

Equation 1: Basic COCOMO equation

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, *Basic COCOMO* is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes. *Intermediate COCOMO* takes these Cost Drivers into account and *Detailed COCOMO* additionally accounts for the influence of individual project phases: (Malevanny, 2005)

There are other additional cost factors proposed by Boehm et al in the COCOMO II model for software engineering cost estimation which includes:

- Product factors*:* This includes reliability, product complexity, database size, required reusability, and documentation matched to life-cycle needs.
- Computer factors: Includes execution time constraints, storage constraints, computer turnaround constraints, and platform volatility.
- Personnel factors: Consist of the capabilities of analysts, application experience, programming capabilities, platform experience, language and tool experience, and personnel continuity.
- Project factors: The set of which is made up of multisite development, software tools used, and development schedule: (Boehm et.al 2000).

Advantages:

- It can generate repeatable estimations.
- It is easy to modify input data and customize formulas.

- It is efficient and able to support different estimations methods.

Disadvantages:

- It is unable to deal with unpredicted situations.
- A mistake in the inputs can generate inaccurate estimation.
- Human experience and speed cannot be easily quantified.

3.2.2 Putnam's model

Lawrence Putnam derives his model based on Norden/Rayleigh manpower distribution and his finding in analysing many completed projects in the 1970s. In this model the association between effort and size is non-linear: (Putnam, 1978). The Putnam model is sensitive to deliver software project on time. According to Putnam model, small additions in the project implementation schedule can result in extensive investments of effort: (Putnam, 2003) The main equation for Putnam's Model is:

$$\frac{B^{1/3} * Size}{Productivity} = (\text{Effort})^{1/3} * (\text{Time})^{4/3}$$

- Size is the product size.
- B is a scaling factor and a function of the project size.
- Productivity is the ability of a particular software firm to produce software of a given size at a particular defect rate.
- Effort is the total effort required for the project.
- Time is the total schedule of project.

$$\text{Effort} = \left[\frac{Size}{Productivity \ . \ Time^{4/3}}\right]^3 * B$$

Equation 2: Putnam model

4 RESEARCH METHODOLOGY

This chapter describes the research approach and the setting for this study. Its objective is to strengthen an understanding of how the research is organized and conducted. This chapter: (a) provide a background to the choice of research methods; (b) describe the selection of research methods and elaborate the research design; (c) explain the data collection; and discuss the validity of the research.

4.1 Choice of the research method

This chapter introduces the research methodology applied in this study. Selecting a right procedure for a research is fundamental to its success. The choice of research method has been done so that it addresses the complex innovative nature of the subject.

A comparative case study approach was chosen because it describes the procedures involved in establishing the relationship and differences between explanatory variables (Pickvance, 2005). This research method emphasized on the explanation of differences and similarities.

The main goal of this research is to analyse how two software cost estimation methods – Top-down method and the Bottom-up method can be compared in relation to the actual user data. This approach strives for a holistic and in-depth analysis of the phenomenon than quantitative research (Yin 1994; Nahar 2000).

Literature review, interview and data collection approach was used to investigate the research question to enhance confidence in the ensuing findings, mitigate the weaknesses of the research method approach which is inherent in many qualitative studies and in so doing, validate the data through cross verification by using data from more than two source ((Webb, Campbell, Schwartz & Sechrest, 1966), (O'Donoghue & Punch, 2003).

An empirical model uses data from previous projects to evaluate the current project, while analytical model, on the other hand, uses formulae based on global

assumptions, such as the rate at which developer solve problems and the number of problems available (Hareton & Zhang, 2003).

4.2 Selection of Research method and data collection

The selections of an appropriate research method hinges on several factors. Some key factors include: the nature of the phenomenon, the state of existing knowledge, and the types of questions to be asked ((Babbie, Survey Research Methods, 1973), (Babbie, 2008), (Dash, 2005)).

For example, different research methods, like action research, grounded theory, case study research, archival analysis have been proposed for conducting qualitative and quantitative research. All these research methods have different techniques for collecting data such as interview, observation, and surveys. The various research methods answer different research questions, and they have different control and time focus.

Both quantitative and qualitative data were collected. As stated earlier, the quantitative data was grounded on three basic data points, i.e. time, size and defect (Humphrey, 1995). While several other interesting data points could have been captured, these three metrics were seen to be the most beneficial for setting some references for other researchers and practitioners.

4.3 Research Design

The main aim of a research design is to ensure that the evidence obtained in data collection and analysis enables the researcher to answer the initial question as unambiguously as possible (Creswell, 2003). It is important because it provides a framework within which the research is conducted and enables both the researcher and subsequent readers of the research to be able to make sense of the study by understanding the role and relevance of the different components of the research.

However, obtaining relevant evidence requires that the researcher specifies the type of evidence needed to answer the research question and evaluate the concept or accurately describe the phenomenon. Failure to have a coherent research design early in the study, may lead to unconvincing answers to the research question and inexact conclusions.

4.4 Limitation of the research Methods

A case study approach has certain limitations that need to be considered. First and foremost, the result of the case study cannot be applied directly to all environments (Yin, 1994). In this research, the case study was an analysis of a single company that has its own specific operational style, target market, location, policy, ethics, and goals. This study cannot be generalized or regarded as flawless and might require more cases with different dependencies to have a comprehensive outcome. Miles and Huberman (1994) have shown that a multiple case research generates more explanatory and generalized outcomes than a single case study which may be applicable to all situations. According to Yin (1994), the choice of a case company is critical because it affects the overall quality of the study.

Furthermore, the amount of the information retrieved can become incredibly large if the method of studying the case is utilized in a wrong way. It could cause difficulty in summarizing and analysing the case. According to Nahar (2000), the research framework, a preliminary interview protocol and a questionnaire guide can be utilized to maintain focus on data collection and to reduce the amount of material to be processed.

Thirdly, the role of the participant in the company or the project to be investigated, also has a significant effect on the quality of the data gathered. This represents one of the biggest challenge of data collection. Top officials in an organization are sometimes too busy or are not willing to give relevant information about their organization because of privacy issues and its accessibility to their competitors.

4.5 Validity and Reliability Measures Taken

To ensure validity and reliability of this research, many measures were applied. This includes.

- Theoretical part of this research (literature review) is based on existing and academically acknowledged theories.
- The case study was studied in two part: The article used as a case study has been reviewed using data collected all through the software development process.

5 CASE STUDY

Two of the most widely used software cost estimation methods; the Bottom-up and Top-Down methods will be compared alongside the user data obtained, to demonstrate if a software development project can be estimated accurately at the beginning of the development or not. A comparative case study approach was chosen for this research.

5.1 Background of Case Study

The data studied for this research was obtained from one of the researches conducted by VTT Technical Research Center of Finland. VTT is regarded as one of Europe's foremost research centre, that endeavours to advance the implementation and commercialisation of research and technology. Through scientific and technological methods, the institution has been able to turn several global challenges and problems into feasible growth for business and society (https://www.vttresearch.com/en/about-us/what-vtt).

The dataset is from a project conducted from the research centre called eXpert. A web-based application for data management is developed by four software engineers and scheduled to be completed in eight weeks. Java application development platform using the latest open-source production tools (eg Eclipse 2.1, www.eclipse.org) as well as configuration management, unit and integration testing tools was used for the development of the application. The development is guided by the Extreme Programming production method, which is thoroughly introduced with tool support in VTT's laboratory facilities. The tools and software used are presented in Table 1 below.

Table 1: Tools and software used:

Item	**Description**
Language	Java (JRE 1.4.1), JSP (2.0),
Database	MySQL (Core 4.0.9 NT, Java connector 2.0.14).
Development Environment	Eclipse (2.1).
SCM	CVS (1.11.2); integrated to Eclipse.
Docs	MS Office XP.
Web Server	Apache Tomcat (4.1).

The schedule, (i.e., from February 3rd, 2003 to March 28th, 2003) and resources for the project are fixed, even though the system requirements are not fully understood at the beginning due to large number of potential users (300+) and their contradicting views. Due to the fixed schedule, all project work is completed at the VTT's workspace with the support of a VTT expert to help with all possible obstacles. Table 2 shows the schedule for each release.

Table 2: Release Schedule

Release number / meeting	**Date**
Steering group kick-off meeting	11.2.2003
SW Release 1	14.2.2003
SW Release 2	28.2.2003
SW Release 3	14.3.2003
SW Release 4	21.3.2003
Steering group meeting II	25.3.2003
SW Release 5 / Final	28.3.2003
Steering group final meeting	15.4.2003

5.2 Data Collection

The obtained data is based on three data points: time, size, and defects. The dataset is arranged around five system releases, each which were tested by 17 customer testers. Activities recorded in the minutes include planning, meeting, coaching, brainstorming, post-mortem, project management, design, pair, and self-programming. The time documented for pair and self-programming are gotten from time in minutes recorded for spike coding, unit testing, coding in Java and Java Server Pages (JSP) and refactoring.

Table 3 below presents the breakdown of effort in minutes used to complete the application development. Each task is organized by the effort accumulated per week and summed up for each release. Release 1, 2 and 3 are each completed after two weeks while Release 4 and 5 are completed after one week. Release 6 (i.e., the final week) is the time scheduled for project delivery.

Table 3: Summary of project hours.

Summary of Project Hours															
Weeks	Planning game	Wrap-up	Meetings	Coaching	Brainstorming	Post mortem	Project management	Design	Miscellaneous tasks	Pre-release testing & bugfix	Pair programming	Self programming	Working minutes	Working hours	Release Total in Details
Week 1	840	0	120	0	0	0	0	480	0	0	0	0	1440	24	
	0	0	120	0	0	0	0	150	0	0	1260	90	1620	27	
	0	60	125	120	0	0	0	0	0	0	1074	115	1494	24.9	
	0	0	0	75	0	0	190	0	0	0	916	70	1251	20.85	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Week 2	0	25	45	0	0	0	415	0	0	0	623	225	1333	22.22	**R1 Total** Minutes: 11725 Hours: 195.417
	0	0	340	0	0	0	100	0	0	0	750	170	1360	22.67	
	0	50	0	45	70	0	35	0	0	0	822	50	1072	17.87	
	0	0	0	15	35	0	15	0	0	0	811	305	1181	19.68	
	0	0	0	0	0	0	232	0	0	0	300	442	974	16.23	
Week 3	800	0	0	0	0	640	185	0	0	0	0	0	1625	27.08	**R2 Total** Minutes: 11381 Hours:
	0	10	34	0	0	0	215	0	767	0	162	155	1343	22.38	
	0	241	0	0	0	0	75	0	0	0	1264	0	1580	26.33	

	0	45	95	0	0	0	106	0	0	0	923	20	1189	19.82	
	0	0	0	60	0	0	80	25	0	0	162	145	472	7.87	
											0	0			
Week 4	0	30	0	0	0	0	120	0	0	0	545	302	997	16.62	
	0	30	0	30	0	0	176	46	0	0	510	165	957	15.95	
	0	70	35	0	0	0	130	50	0	0	626	355	1266	21.1	
	0	0	8	0	0	0	260	0	150	534	469	306	1727	28.78	
	0	0	0	0	0	0	0	0	225	0	0	0	225	3.75	
Week 5	1190	0	0	0	0	440	140	0	0	0	0	0	1770	29.5	**R3 Total** Minutes: 11622 Hours: 193.7
	0	97	0	0	0	0	135	0	0	0	632	148	1012	16.87	
	0	30	0	0	0	0	30	0	0	0	530	260	850	14.17	
	0	105	0	0	0	0	213	0	0	0	565	56	939	15.65	
	0	10	0	0	0	0	10	0	0	0	255	410	685	11.42	
Week 6	0	175	0	99	0	0	145	0	0	0	820	223	1462	24.37	
	0	155	0	0	0	0	115	0	0	0	923	185	1378	22.97	
	0	10	0	30	0	0	45	0	0	0	632	435	1152	19.2	
	0	120	0	0	0	0	160	0	0	284	590	110	1264	21.07	
	0	170	0	0	0	0	5	0	300	635	0	0	1110	18.5	
Week 7	1050	0	0	0	0	240	140	0	0	0	0	0	1430	23.83	**R4 Total** Minutes: 6644 Hours: 110.733
	0	0	0	38	0	0	133	0	0	0	926	267	1364	22.73	
	0	90	0	0	0	0	20	0	240	0	1369	181	1900	31.67	
	0	65	0	0	0	0	120	0	0	910	130	205	1430	23.83	
	0	50	0	0	0	0	65	0	0	405	0	0	520	8.67	
Week 8	517	69	0	240	0	220	140	0	0	0	411	0	1597	26.62	**R5 Total** Minutes: 5657 Hours: 94.2833
	0	190	445	25	0	0	95	0	20	0	422	146	1343	22.38	
	0	15	0	0	0	0	145	0	335	0	240	580	1315	21.92	
	0	10	0	0	0	0	181	0	447	325	139	300	1402	23.37	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Week 9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**R6 Total** Minutes: 2264 Hours: 37.73
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	315	0	0	0	0	0	153	0	180	0	656	10	1314	21.9	
	0	0	0	0	0	0	160	0	420	310	0	60	950	15.83	
						0									
Summary	4712	1922	1367	777	105	1540	4684	751	3084	3403	20457	6491			

Table 4 below, summarizes the effort accumulated per release while Table 5 presented the data per release in hours. These tables gave a clearer picture of what activities got more effort per release. For example, the highest effort for coding was done during the first release. The graph below also displayed the visual representation of the trends, relationships and dependencies of the variables.

Table 4: Summary of Effort used for each release in minutes.

	Planning game	Wrap-up	Meetings	Coaching	Brainstorming	Post mortem	Project management	Design	Miscellaneous tasks	Pre-release testing & bugfix	Pair Programming	Self programming
R1 effort	840	135	750	255	105	0	987	630	0	0	6556	1467
R2 effort	800	426	172	90	0	640	1347	121	1142	534	4661	1448
R3 effort	1190	872	0	129	0	440	998	0	300	919	4947	1827
R4 effort	1050	205	0	38	0	240	478	0	240	1315	2425	653
R5 effort	517	284	445	265	0	220	561	0	802	325	1212	1026
R6 effort	315	0	0	0	0	0	313	0	600	310	656	70
Total	4712	1922	1367	777	105	1540	4684	751	3084	3403	20457	6491

Table 5: Summary of Effort used for each release per hour.

	Planning game	Wrap-up	Meetings	Coaching	Brainstorming	Post mortem	Project management	Design	Miscellaneous tasks	Pre-release testing & bugfix	Pair Programming	Self programming
R1 Effort/h	14	2.3	12.5	4.3	1.8	0	16.5	10.5	0	0	109.2	24.5
R2 Effort/h	13.3	7.1	2.9	1.5	0	10.7	22.5	2	19	8.9	77.8	24.2
R3 Effort/h	19.8	14.5	0	2.2	0	7.3	16.6	0	5	15.3	82.6	30.4
R4 Effort/h	17.5	3.4	0	0.6	0	4	8	0	4	21.9	40.4	10.9
R5 Effort/h	8.6	4.7	7.4	4.4	0	3.7	9.4	0	13.4	5.4	20.2	17.1
R6 Effort/h	5.3	0	0	0	0	0	5.2	0	10	5.2	10.9	1.2
Total Effort/h	78.5	32	22.8	13	1.8	25.7	78.2	12.5	51.4	56.7	341.1	108.3

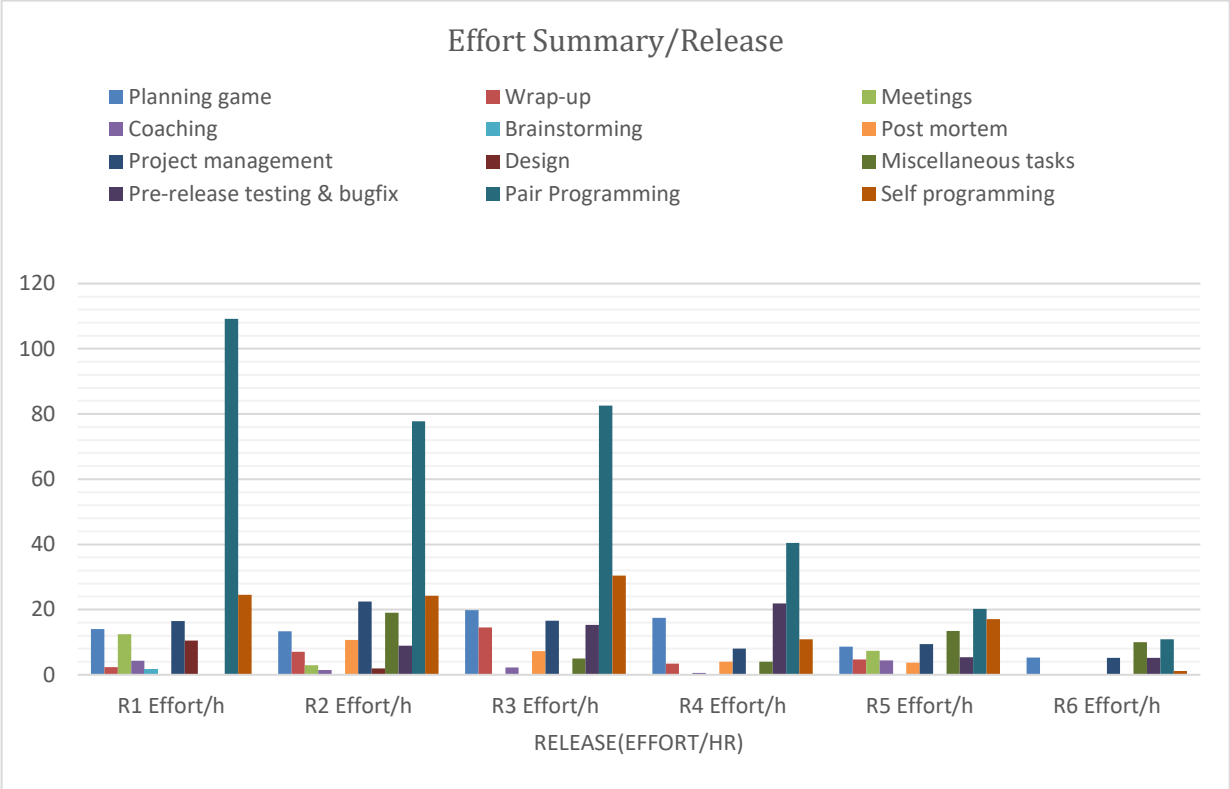

Figure 1 : Graphical representation of effort in each release

Overall, 7698 Lines of Code and 820 hours were used for the project. While several points of comparison were established, the more interesting and noticeable, is that the effort required for coding costs 54.67% of the total development effort while planning, which is also very important, takes 12.68% of the total effort.

Table 6: Percentage of effort used in hours.

	Programming Part (in hours)	Project Management Part (in hours)	Planning part (in hours)	Other part of the project (in hours)	Total
R1	133.7	37.2	14	10.5	195.4
R2	101.8	33.9	24	30	189.7
R3	112.9	33.3	27.2	20.3	193.7
R4	51.3	12	21.5	25.9	110.7
R5	37.3	25.9	12.3	18.8	94.3
R6	12.1	5.2	5.3	15.2	37.7
Total	449.1	147.6	104.2	120.6	821.6

	Programming Part (in hours)	Project Management Part (in hours)	Planning part (in hours)	Other part of the project (in hours)	Total
R1	68.43%	19.04%	7.16%	5.37%	100.00%
R2	53.68%	17.88%	12.65%	15.79%	100.00%
R3	58.29%	17.20%	14.03%	10.49%	100.00%
R4	46.33%	10.85%	19.42%	23.40%	100.00%
R5	39.56%	27.49%	13.03%	19.92%	100.00%
R6	32.07%	13.83%	13.91%	40.19%	100.00%
Total	54.67%	17.96%	12.68%	14.68%	100.00%

5.2 Data Usage

Many datasets from this project by VTT Technical Research Center of Finland was made available and analysed for this research. Some of the datasets were not used but are included in the appendix section of this report because they provide detailed information about some of the variables or dataset used in this analysis.

The first dataset in the appendix section is the detailed summary table, that shows the distribution of effort during development. This table provide information like the Line of Code (LOC), Team productivity/hour (i.e., LOC/hr), code integration, average time between integration, number of user stories implemented during each release, post release defects and percentage of pair programming used.

The second table is the Release history with user requirements. It gives some information about the timeline of each user-story with the estimated and actual resource used. The other table is the "Defects" table. It provided more information about the defects detected during each release and when it was fixed.

The table below provide the list of dataset and how well it was used in this research.

Table 7: Summary of dataset usage

Document Identification	**Insight**	**Usage of dataset**
Project Plan	This document includes change history, breakdown of release content, release Schedule, references, tools and software used.	Yes. Most part of this document were used for this research
Summary of Projected Hours	This was the document that has most of the data used for the research. It includes the effort chart, description of data used for analysis (Legend), chart summary, pair vs solo programming chart, summary of project	Yes. Most part of this document was used for the research
Record of defects	Records of defects encountered during development. The records include the defect date, during what release, explanation of what was done to rectify defect, severity and time required to fix defect.	No. This document was added to the appendix but the data in it was not used in this research
Change history	History of all the changes made during the development. This doc has the version, date of change, comment regarding the change	No. This document was added to the appendix, but the records was not used in the research
Resource Estimation	This document explained calendar time vs programmer time. It explained the concept of pair programming. It also gives a description of how much time is dedicated for each release with respect to customer stories	

6 EMPRICAL ANALYSIS

This chapter represents the empirical analysis of this study. It starts with the description of all the user stories described in the research, explanation of the amount of effort used for the project, estimate developed if the project would have been estimated using the bottom-up and top-down estimate and the comparison between the three estimates. Three primary empirical contributions or deductions (PEC) will be developed from what is noticeable from the comparison.

Each task is organized by the effort (in hours) accumulated per week and summed up for each release. Release 1, 2 and 3 are each completed after two weeks while Release 4 and 5 are completed after one week. Release 6 (i.e., the final week or project closing and presentation week) is the time scheduled for project delivery.

Each release is allocated with some use-cases that need to be completed. The user stories are described below.

Release 1 was scheduled to be completed in 2 weeks and is the start of the project. The developers are expected to complete each user story within the time frame allocated within each release. User story 1 – 5 are tasks allocated to this release. These are:

- User story 1: User must be able to create workspace. This task includes initial database and software installations.
- User story 2: User must be able to create virtual folder.
- User story 3: User must be able to create virtual file.
- User story 4: User must be able to browse workspaces and folders.
- User story 5: User must be able to open virtual file.

Some hours are allocated for tasks like the project management and planning.

Release 2 is planned to be completed in 2 weeks and contain user story 6 to 17. This includes:

- User-Story 6: User must be able to delete Workspace, Folder, File
- User-Story 7: User must be able to assign owners to workspace, folder, file.
- User-Story 8: User should be able to sort list.
- User-Story 9: User should be able to update workspace, folder, and file.
- User-Story 10: User should be able to view all file links.
- User-Story 11-12: Bug detection for release 1
- User-Story 13-14: Release 1 bug fixes and enhancement
- User story 15: User should be able to search for keywords and words in description.
- User story 16: User should be able to update sorted list in User story 8.
- User story 17: User should be able to use keywords to find files, folders, and workspaces.

Some hours are allocated for tasks like minor bug fixes, pre-release tests, code review, server environment updating and testing, task estimation and audit.

Release 3 is scheduled to be completed in 2 weeks and contains user story 18 – 26. These stories are:

- User -story 18: Release 2 bug detected and fixed with enhancements.
- User story 19: Icon for files and folder created.
- User story 20: Administration password for maintenance
- User story 21: Find related function for item.
- User story 22-23: Top 10/100 of the most/least accessed
- User story 24-25: Copy/move function for folders. Password handling required refactoring.
- User story 26: User can view and choose from list of existing keywords on new edit, search forms.

During this release, one hour per day was set aside for management and wrap-up.

Release 4 is planned to be completed in 1 week and contain user story 27 – 31. These stories are:

- User story 27: Release 2 bug detected and fixed.
- User story 28: Release 3 enhancements: mandatory fields, highlight open workspace, CSS usage, password for editing, Item-List refactoring, time stamps hh:mm, show folder into for new/edit/delete-functions.
- User story 29: User can copy/move selected folders and resources, enhancements. Refactoring view .jsp.
- User story 30: User manual/helps created.

- User story 31: SW design documentation created but not fully implemented.

Release 5 is the final part of the project. It is scheduled to be completed in 1 week and contain user story 32 – 35. The presentation and delivery of the project happened during the next week and might be included in this release.

- User story 32: Release 4 bug detected and fixed.
- User story 33: Release 4 enhancements
- User story 34: Admin tool
- User story 35: Playground

Delivery of the project could be regarded as user story 36 and some hours are allocated to this.

The next part of this empirical analysis states the effort (measured in hours), allocated to the user stories, and summed up for each release according to each estimation method examined.

6.1 Baseline Estimation

The baseline estimation is the effort, in hours allocated to each release according to the actual project. This include hours that might have not been set aside or just described as miscellaneous. In this project, some tasks allocated to a specific release are completed in the other release because of some unforeseen circumstance. For example, User story 35 was postponed, because the server was not delivered in time. In release 4, more testing was conducted, because some unexpected bugs were detected and had to be fixed before the development process could continue.

The breakdown of the project demonstrates that although all the tasks in the project can be broken down and scheduled, there are so many unexpected failures, bugs, incidents, events that could alter the project schedule and delivery.

The table below displays how much effort it takes to complete each user story and the reason why that amount of effort was used. There are also comments in the description column, stating what happened or how many hours were added to a task.

Table 8: Baseline data with release history, user story and effort allocated.

Release	User story number or tasks	Description *Remarks*	Actual resource use
Release 1 (2 weeks)	Project mgt and other	Contains time which cannot be recorded to tasks.	35 h
	Story #1	User must be able to create workspace. *This task includes initial database and software installations*	51 h
	Story #2	User must be able to create virtual folder.	63 h
	Story #3	User must be able to create virtual file.	10 h
	Story #4	User must be able to browse workspaces and folders.	6 h
	Story #5	User must be able to open virtual file.	6 h
	Planning day	Includes task estimation (and in following releases auditing and post mortem of previous release)	24 h
	Total		**195 h**
Release 2 (2 weeks)	Project mgt and other	Contains time which cannot be recorded to tasks.	34 h
	Story #6	Delete Workspace, Folder, File	14,5 h
	Story #7	Owner to workspace, folder, file	7 h
	Story #8	Sorting lists	8 h
	Story #9	Update Workspace, Folder, File	10 h
	Story #10	View All File Links	4 h
	Story #11	*Bug Report Function* *Postponed to release 3.*	
	Story #13	Release 1 bug fixes and enhancements	8,5 h
	Story #15	Search for keywords and words in descriptions	27 h
	Story #16	Last Updated field *(** transferred 1h to story 8)*	3 h
	Story #17	Keywords to files, folders, and workspaces	14 h
	Pre-release tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews.	28 h
	Planning day	Includes task estimation and auditing and post mortem of previous release.	32 h
	Total		**190 h**
Release 3 (2 weeks)	Project mgt and related	Estimated 1 hours per day for management tasks and wrap-up. *Estimated increase in wrap-up on later releases.*	33,5 h
	Planning day	Includes task estimation and auditing and post mortem of previous release. *Post mortem was mistakenly left out, removed hours from (other unexpected).*	27 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews. *"Misspent" time is attempted to capture to here instead of project management/wrap-up. Code reviews (15h) moved here. Pre-release test & fix estimate (16h). Other unexpected 4h*	20,5 h

		No time was left for code reviews. Miscellaneous (5h): UI enhancements.	
	Story #11	Bug report function	6,5 h
	Story #18	Release 2 bug fixes and enhancements	41 h
	Story #19	Icons for folders and resources	2 h
	Story #20	Administration password for maintenance	0,5 h
	Story #21	Find related –function for items	7,5 h
	Story #22	Top 10/100 of most/least accessed resources/folders. *Java date handling caused problems as well as somewhat more complex SQL. Hit collecting wasn't as easy as expected (problems with Netscape and JavaScript), link handling was improved.*	19 h
	Story #24	Copy/Move function for folders. *Password handling required refactoring, underestimated complexity regarding to functions itself.*	18,5 h
	Story #26	User can view and choose from list of existing keywords on new, edit, search forms. *Changes to server components were under estimated. JavaScript connectivity between windows was more difficult than expected.*	18 h
	Total		**194 h**
Release 4 (1 week)	Project mgt and related	Estimated half of the actual effort in R3. *Drop from 7 to 3 days, estimated one day too much.*	12 h
	Planning day	Includes release planning and post mortem of previous release	21,5 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews. *Pre-release test & fix estimate (8h). No miscellaneous or code reviews expected.* *Tasks were completed early in general and left over time was spent testing and enhancing functions. Thursday testing left some bugs which were fixed on Friday which made for almost extra day. Miscellaneous 4h of improving pictures.*	26 h
	Story #27	Release 3 bug fixes.	6 h
	Story #28	Release 3 enhancements: *mandatory fields, highlight open workspace, CSS usage, password for editing, ItemList-refactoring, time stamps hh:mm, show folder into for new/edit/delete-functions*	15 h
	Story #29	User can copy/move selected folders and resources, enhancements. Refactoring view.jsp	22 h
	Story #30	User manual / helps	6 h
	Story #31	SW Design Documentation will not be fully implemented in R4.	2,5 h
	Total		**111 h**
Release 5 - Part 1 (1 week)	Project mgt and related	Estimated more than R4 because of project ending and steering group meeting.	26,5 h
	Planning day	Includes release planning and post mortem of previous release. Little new functionality so expected less than R4.	12,5 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews. *Pre-release test & fix estimate (10h). No miscellaneous or code reviews expected. Possible left over time is spent here.*	15 h
	Story #30	Helps *Largely underestimated*	5 h
	Story #31	Documentation	5 h

		Part of documentation was moved to R6 (design doc)	
	Story #32	R4 bug fixes *It was expected that some JavaScript would need to be done but wasn't.*	1 h
	Story #33	R4 enhancements	15 h
	Story #34	Admin tool	16 h
	Story #35	Playground *Postponed to R6 because server was not delivered in time.*	0 h
	Total		**96 h**
Release 5 - Part 2	Planning	Planning of final release	5,5 h
	Story #35	Playground *Server was not purchased early enough for application to be deployed.*	-
	Story #31	Documentation *(design doc)* *Time recorded into misc tab in project time sheet.*	7 h
	Story #36	R5 bug fixes *Mainly 1h and 2h estimated, some were done in 10 minutes.*	7,5 h
	Story #37	R5 enhancements	3,5 h
	Project post-mortem	*Was forgotten and not recorded, estimated spent 12h*	
	Miscellaneous	*Possible last day stuff, not estimated*	
	Pre-release testing	*Was left un-estimated*	5 h
	Project management and related	Project management and shutdown	5 h
	Total		**38 h**

6.2 Bottom-up Estimation

Bottom-Up Estimation is mostly implemented once detailed information about the project is made available, thereby making it easier to create a work breakdown structure. A Work Breakdown Structure (WBS) is the process of communicating all the work that needs to be carried out on a project, broken down into smaller work packages and documented in a ranking structure: (Kruchten, 2007).

In this estimation method, effort is estimated for each work package which are generated by experts and added up to arrive at a total estimate. It helps avoid cost and payment error in fixed price contracts. It is mostly used for budgeting, scheduling, fund timing and resource requirement.

Some types of Bottom-up estimation

- Single Point Estimation. This method uses a figure or number to specify certain information like date, time, days e.t.c. For example, how long will "Task A" take to complete can be specify with 18 months, 6 days, 4 hours e.t.c.
- Three-Point Estimation: Three different figures are used to specify certain details. These include the optimistic (a), most likely (m) and pessimistic (b). This method is rarely used, because it makes the estimator look incompetent and does not provide the project manager the right information or data needed for the project. These figures are used to calculate the Activity Time Calculation which is:

$$\text{Activity Time Calculation (Te)} = \frac{a+4m+b}{6}$$

Equation 3: Activity Time Calculation

The data collected from the study used for this project, shows that each release is broken distinct task which are represented by user story. This process is the same as the bottom-up method. Every task is broken down into smaller part (i.e., user story in this research) and effort in hours are allocated to each. The bottom-up estimate is only an estimate and might not include some of unplanned incident or events encountered in the project. Some effort time might be allocated for emergency, but it is just an estimate that might either be more or less than the allocated time.

The table below the estimated time for the project using the bottom-up method.

Table 9: Bottom-up estimate with release history broken down to user stories and effort allocated.

Release	**Story#**	**Description**	**Estimated resource use**
Release 1 (2 weeks)	Story #1	User must be able to create workspace. Task include creation of database and software installation.	96 h
	Story #2	User must be able to create virtual folder.	48 h
	Story #3	User must be able to create virtual file.	12 h
	Story #4	User must be able to browse workspaces and folders.	12 h
	Story #5	User must be able to open virtual file.	12 h

	Planning day	Includes task estimation (and in following releases auditing and post-mortem of previous release)	12 h
	Total		**192 h**
Release 2 (2 weeks)	Project management and other	Contains time which cannot be recorded to tasks.	28 h
	Story #6	Delete Workspace, Folder, File	21 h
	Story #7	Owner to workspace, folder, file	6 h
	Story #8	Sorting lists	7 h
	Story #9	Update Workspace, Folder, File	6 h
	Story #10	View All File Links	6 h
	Story #11	*Bug Report Function* *Postponed to release 3.*	*4 h*
	Story #13	Release 1 bug fixes and enhancements	20 h
	Story #15	Search for keywords and words in descriptions	20 h
	Story #16	Last Updated field	4 h
	Story #17	Keywords to files, folders, and workspaces	18 h
	Pre-release tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews.	28 h
	Planning day	Includes task estimation and auditing and post-mortem of previous release.	24 h
	Total		**192 h**
Release 3 (2 weeks)	Project management and related	Estimated 1 hours per day for management tasks and wrap-up.	36 h
	Planning day	Estimation and auditing and post-mortem of previous release.	27 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews.	*35 h*
	Story #11	Bug report function	8 h
	Story #18	Release 2 bug fixes and enhancements	35 h
	Story #19	Icons for folders and resources	5 h
	Story #20	Administration password for maintenance	1 h
	Story #21	Find related function for items	7 h
	Story #22	Top 10/100 of most/least accessed resources/folders.	12 h
	Story #24	Copy/Move function for folders.	14 h
	Story #26	User can view and choose from list of existing keywords on new, edit, search forms.	12 h
	Total		**192 h**
Release 4 (1 week)	Project management and related	Estimated half of the actual effort in R3.	17 h
	Planning day	Includes release planning and post mortem of previous release	24 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews.	8 h
	Story #27	Release 3 bug fixes.	12 h

	Story #28	Release 3 enhancements: *mandatory fields, highlight open workspace, CSS usage, password for editing, ItemList-refactoring, time stamps hh:mm, show folder into for new/edit/delete-functions*	17 h
	Story #29	User can copy/move selected folders and resources, enhancements. Refactoring view.jsp	18 h
	Story #30	User manual / helps	4 h
	Story #31	SW Design Documentation will not be fully implemented in R4.	2 h
	Total		**96 h**
	Project management and related	Estimated more than R4 because of project ending and steering group meeting.	20 h
	Planning day	Includes release planning and post mortem of previous release. Little new functionality so expected less than R4.	12 h
	Pre-release and other miscellaneous tasks	Minor bug fixes, server environment updating and testing, pre-release tests, code reviews.	14 h
Release 5 (Part 1 and 2) - First week for concluding part of the project and 2nd week for project delivery	Story #30	Helps	3 h
	Story #31	Documentation	11 h
	Story #32	R4 bug fixes	5 h
	Story #33	R4 enhancements	15 h
	Story #34	Admin tool	14 h
	Story #35	Playground	2 h
	Planning	Planning of final release	5,5 h
	Story #35	Playground	2 h
	Story #31	Documentation	6 h
	Story #37	R5 enhancements	5 h
	Project management and related	Project management and shutdown	4 h
	Total		**132 h**

PEC 1: The bottom-up estimate is the closest estimate to the baseline estimate. They both have almost the same features and estimate, except that the effort hours allocated for non-technical tasks and unexpected events in the bottom-up estimate are insufficient or inaccurate.

6.3 Top-down Estimation

Top-Down Estimation method is the process where the total cost or effort required for a project is determined at the beginning of the project. The smaller

part of the project is subsequently broken down with reference to the total calculated or estimated at the beginning of the project: (Nguyen, 2010). This method is usually implemented by senior management based on the general knowledge accessible about the project. It is supported by experience and expert and rely on historical data of old projects or projects completed by competitor. It is a quick and easy method and ignore technical details of project which possibly yield inadequate result since it is no specific metric to measure programmer efficiency. It could be used for fund requirement, resource capacity planning and feasibility study (Iqbal et.al., 2017).

Some types of top-down estimation

- Consensus Method: relies on the experience of several senior managers to improve the accuracy of the estimate. Could be regarded as pooled experience.
- Ratio Method: relies on the fundamental project attributes like size, cost/feature, and duration.
- Apportion Method: Calculate the cost of individual tasks as a percentage of the total cost. For example, a project (100%) can be divided to the following: Design (15%), Programming (40%), Test (35%) and Documentation (10%).
- Learning Curve: accounts for the fact that each time a task is repeated, it will take less time to complete. The concept of pair programming is used to describe this and shows how effort could be maximized when two programmers do a task together or when a repetitive task is done again. With pair programming, the quality of the result is better with less calendar time, but the person-hours increases.

Estimating the effort required for the project, using the top-down method can would require some historical data and reference to past projects. Since this information cannot be retrieve, the number of software developers, hours allocated for full time work per week, and number of weeks required to complete the project can be used to calculate the total effort required for the project.

In this project, four developers are assigned to complete the project. They will be working full-time, for 8 weeks and the 9th week will be dedicated for closing and delivering the project. Full time for the project for each developer is 24 hrs/week (i.e., 6 hours/day for 4 days a week). This means that 96 hours of effort is needed per week for all the four developers. Some tasks (like coding

and testing) can be completed using pair programming while other task like lunch break, team meetings can only be completed individually.

For one programmer (Full time – 6hrs per day = 24hrs required per week.)

- Estimated break/relaxation time – 1 hr per day = 4hrs per week
- Estimated meeting/Planning – 0.75 hr per day = 3 hrs per week
- Estimated coding time – 4.25 hrs per day = 17 hrs per week

Since there are no data that specifies which task will be completed individually or with pair programming, it can be assumed that 20% of the coding task will be completed by pair programming but will not be considered in this estimation. It is also good to note that some codes will be used in some repetitive tasks, which will reduce the effort time.

The table below the estimated time for the project using the top-down method.

Table 10: Top-down method estimated as the developer's total effort per week.

Release	Week	Developer effort/week (in hours)	Total effort / week (in hours)
	1	24	96
	2	24	96
Release 1	**Total**	**48**	**192**
	3	24	96
	4	24	96
Release 2	**Total**	**48**	**192**
	5	24	96
	6	24	96
Release 3	**Total**	**48**	**192**
Release 4	7	24	96
	Total	**24**	**96**
	8	24	96
	9	12	48
Release 5	**Total**	**36**	**144**

PEC 2: The top-down estimate is just an approximate that is likely flawed. Even though the estimate might be close to the baseline estimate, it ignores all the technical and non-technical details, unforeseen events, and work breakdown structure of both the baseline and bottom-up estimate.

6.4 Comparison

In this section, the result from the three estimates above are compared. The comparison table will include the total effort hours obtained for the baseline estimation for each release in comparison with the bottom-up and the top-down estimates.

Table 11: Comparison of baseline, bottom-up and top-down estimate

Release	Baseline Estimate (Effort in hours)	Bottom-up Estimate (Effort in hours)	Top-down Estimate (Effort in hours)
Release 1	195	192	192
Release 2	190	192	192
Release 3	194	192	192
Release 4	111	96	96
Release 5	134	132	144

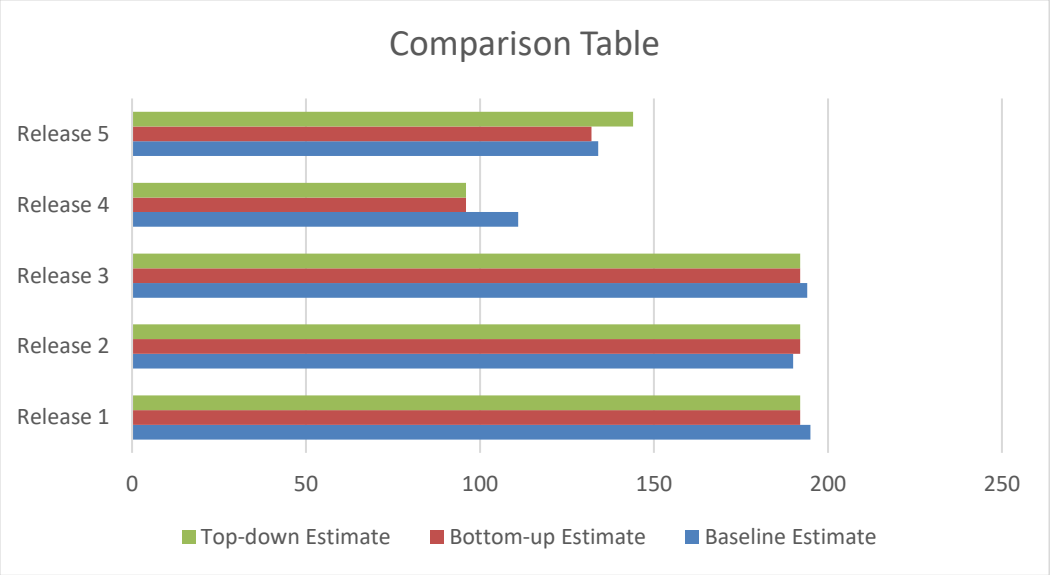

Figure 2: Graphical representation of effort in each release

The effort estimated during the project (i.e., 195 hours) differs from the estimate obtained for both bottom-up and top-down methods. It is coincidental that the effort was required for both the bottom-up and top-down estimates (both 192 hours). Comparing the baseline and bottom-up estimate tables in section 6.1 and 6.2, more effort was allocated to planning and other non-programming

tasks in the baseline estimate at the beginning of the project while there was no time allocated for these tasks in the bottom-up estimate. This process planning reduced the actual amount of effort time required for coding. For example, project management, planning, taking breaks and other non-coding task under release 1, accounted for 59 hours in the baseline estimate and just 12 hours in the bottom-up estimate. This was evident because the coding task for the user stories were completed within 136 hours in the actual project but was predicted to be completed in 180 hours using the bottom-up method. The top-down estimate remains the same per week throughout the project. (i.e., 4 developers working for 24 hours per week requires 96 effort hours per week and 192 hours of effort over 2 weeks). The top-down estimate is a quick way to make an estimate and could be use as a guide of the total effort hours required for the project.

> PEC 3: The more effort hours allocated to planning and project management tasks at the beginning of a software project decreases lag time, identify possible risk, prioritize tasks, and reduces the time spent on the actual coding.

PEC 3 shows that planning, project management and other non-coding task should be given more attention and allocated enough time at the beginning of the project, because it could act as a guide and reduce the time for coding or doing the technical task.

The same pattern, where the more planning done reduces the coding time can also be noticed for data recorded in release 2. Furthermore, user story 11, "Bug Report Function" was estimated to be completed in release 2 but was actual moved to release 3. The 4 hours allocated to this user story under the bottom-up estimate are used for the non-coding task in the baseline estimate.

The effort estimated to complete some user stories in the 3rd release differs to the actual effort used to complete the task. For example, it takes 41 hours to fix the bugs in release 2, but 35 hours was estimated to complete this task. For the first time in the project, the estimated efforted required (192 hours) to complete the task in this release is lesser than the actual time (194 hours) it took to complete the tasks.

PEC 4: Testing and bug fixing are important and delicate part of developing a software. The more effort hour set aside for these tasks in a software project, the likelihood it is for the project to completed and accepted by the customer.

One aspect of a software development project is the little time allocated for testing and bug fixing. While some stakeholders just do not see the reason why they should allocate a lot of time to bug fixing, most developers feel they are good enough that they make little or no mistake while coding and would not require so much time fixing bug. Another reason could be that the software just does not work after different developer must have worked on smaller part of the project. It is necessary that more time should be allocated for testing and bug fixing, even when an experienced developer is involved, so that more testing can be conducted before the software is delivered for use. This is critical for applications that are life threatening if it is not tested extensively and bugs fixed before considered appropriate for use. Testing also help to detect any security vulnerabilities that could cause harm or attack.

Having spent 6 weeks in the project, it could be noticed that more time are spent on pre-release testing, miscellaneous tasks, and enhancement. The predicted effort time for these tasks is 37 hours while 47 hours were used. It is also noticed that the more progress made in the project, the less time spent for planning and project management. For example, it is predicted that 41 hours will be required for planning and managing the project, but only 37.5 hours were used.

The effort required for both estimates in the final release are almost the same. Though pre-release testing (5 hours) was not estimated at all, using the bottom-up method, but over 13.5 hours are estimated to be used for fixing bugs detected in this release. Only 7.5 hours was literally used.

Overall, 824 effort hours was used to complete the projected. The bottom-up method estimated using 804 effort hours and the top-down method estimated 816 hours.

PEC 5: The difference between the baseline estimate obtained from a software project and estimates calculated by various estimation methods at the beginning of the project is insignificant and can be likely ignored if the project is small.

PEC 5 shows that the bigger the software project or the longer the delivery time of a project is, the more difficult it is to estimate the effort required to complete the project with either top-down or the bottom-up estimation method. For example, assuming the data used for this study was to be for one year, it will be impossible to use the top-down method by just multiplying the number of developers with the number of hours required per week by 52 weeks.

Using this method, would yield a total of 4992 effort hours (i.e., 96 hrs x 52 weeks) required to complete the project. The problem with this estimate is that so many uncertainties (some manageable and others unexpected and uncontrollable) like holidays, sick days, trainings, developer quitting, developer productivity for a big project, hardware and software problems, mentoring, requirement changes, natural disaster and many more are not considered. These uncertainties could lead to overestimation or underestimation, which would have an impact on the success of the project.

6.4 Summary of PECs

Five primary empirical conclusions were formed from the data used for research and the data estimated. These primary empirical conclusions are summarized in the table below.

Table 12: Summary of the primary empirical conclusions from the analysis.

PECs	**Summary of PECs**
PEC 1	The bottom-up estimate has almost the same features and estimate as the baseline estimate except it is difficult to accurately estimate the unpredictable events and some non-technical tasks.
PEC 2	The top-down estimate is a quick estimate used to predict what the total project would cost. It is generally undependable and should only be considered for small projects or for project bidding.
PEC 3	At the beginning of the project, more effort time should be set aside for project planning and management, to help create a clear pathway for the structural and technical definition of the project.
PEC 4	Though it is difficult to predict all the events that could happen in a project, enough time should be allocated for bug fixing and testing because it is a necessary and need to be clearly monitored for the success of the project.
PEC5	The inaccuracy in the estimate generated for a small project is generally negligible, because there is the possibility that most estimation method will derive estimates that are almost the same.

7 DISCUSSION

This chapter present the five primary empirical conclusions developed in the previous chapter and attempt to correlate it with theoretical concept of this study.

7.1 Implications for practice

Although, various estimation methods and models have been developed, it is still very challenging for estimators to decide which one is the best. Some estimates are simple to develop but extremely inaccurate while others that seems almost correct, are burdensome to create.

As stated in PEC 1, the bottom-up estimate involves breaking down the projects to small bits to find the total resources, cost or effort required to complete the project. It is usually one of the likely accurate estimates since it takes in consideration almost all the details except the unexpected events that could cause the failure of the project. For example, there is no way an estimate created sometime around January 2020 that has enough time assigned to all the tiny bits of the project, would have allocated enough time for the unexpected lock down that happened due to the Covid-19 pandemic. This only shows that unpredicted events could have an adverse effect on an estimate, no matter how accurate it is.

PEC 2 explains how the top-down estimate is just an approximate. It should only be used in situations where there is the need to have a quick idea of how much a project would cost. This estimate is used to bid for projects, manage a very small project that is not complex and does not require extensive details.

Although, the task in PEC 3 and PEC 4 (planning, bug fixing and detailed testing) are delicate and important measure for the success of a software project, it is given very little regard in PEC 2 and cannot be accurately estimated in PEC 1.

As stated by PEC 5, the choice of the estimation method chosen to estimate a small project might not matter much, since they could generate almost the same estimate, it is worth noting that other factors like experience and skill of the developers assigned to complete the project, unpredicted situations, and ever-changing requirements could still alter the estimate of the project.

7.2 Implications for research

The goal of this section of the thesis is to balance how the five empirical conclusions contributes to scientific knowledge.

Table 13: Theoretical contributions of the study

PECs	Description	Scientific Novelty
PEC 1	The bottom-up estimate is the closest estimate to the baseline estimate. They both have almost the same features and estimate, except that the effort hours allocated for non-technical tasks and unexpected events in the bottom-up estimate are insufficient or inaccurate.	Corresponds with existing knowledge and research. (Boehm, 2001)
PEC 2	The top-down estimate is just an approximate that is likely flawed. Even though the estimate might be close to the baseline estimate, it ignores all the technical and non-technical details, unforeseen events, and work breakdown structure of both the baseline and bottom-up estimate.	Corresponds with existing knowledge and research (Leung & Zhang, 2001)
PEC 3	The more effort hours allocated to planning and project management tasks at the beginning of a software project decreases lag time, identify possible risk, prioritize tasks, and reduces the time spent on the actual coding.	Corresponds with existing knowledge and research.
PEC 4	Testing and bug fixing are important and delicate part of developing a software. The more effort hour set aside for these tasks in a software project, the likelihood it is for the project to completed and accepted by the customer.	Corresponds with existing knowledge and research. (McConnell, 2006)
PEC 5	The difference between the baseline estimate obtained from a software project and estimates calculated by various estimation methods at the beginning of the project is insignificant and can be likely ignored if the project is small.	Contradicts existing knowledge and research

8 CONCLUSIONS

This chapter present the conclusion for the study. These conclusions include the answer to the research question and general insight on what was learnt during the study, limitations of the study and new research areas that could create future research opportunities.

8.1 Answer to research questions

Estimating the cost of a software project at the beginning of a software project, continually remains a challenge. Commenting on the issue, Brooks, in his journal, "*Great Challenges for Half-Century-Old Computer Science*" labels it as one of the three big challenges of computer science practice:

> *Given specific functional, reliability, and performance specifications for a software system, we do not yet know how to estimate the effort required building it. The challenge is to make software engineering as predictable a discipline as civil or electrical engineering. I still do not expect any radical breakthrough, any silver bullet, to solve this problem. But the accretion of many contributions has already made much progress, and I believe continued careful research, ever validated by real practice, will bring us to that goal* (Brooks, 2003)

The objective of this thesis is to analyse the challenges in software project cost estimation. To establish this, three sub questions were formulated:

- What is the best estimation method for any software project?
- What are the key reasons for cost overrun in developing large software?
- Is it possible to estimate a software project, by using the Bottom-Up or the Top-down method alone?

To answer the research question above, a detailed literature review was conducted on estimation methods and challenges encountered in choosing one of them.

The empirical findings suggest that none of the estimation methods can be completely regarded as the best with the highest degree of accuracy. Several reasons are identified as being responsible for this (Linda, 2006). This includes.

- Software development process is moving so fast that it becomes almost impossible to develop a fit one all type cost estimation method.
- Furthermore, changing requirements and unrealistic expectation from the customer are the some of the reasons why it is difficult to create an estimate at the beginning of a project and stick to it all through project.

For example, a software development team might submit a lower bid just to win the bid. During development, the firm then come up with different ways to extract more money or to cut corners in areas of functionality, testing, reliability, etc.

It is recommended to use different estimation methods, because each one of them have their advantages and disadvantages. The estimates developed, should also be updated often throughout the project. (Boehm, 2001). For example, with an innovative idea generated, a good choice can be, starting with a top-down estimate to give a quick view of the total effort hours needed to develop a schedule and budget. Later in the project, a bottom-up estimate can be generated by modifying the schedule and budget and rectifying the difference.

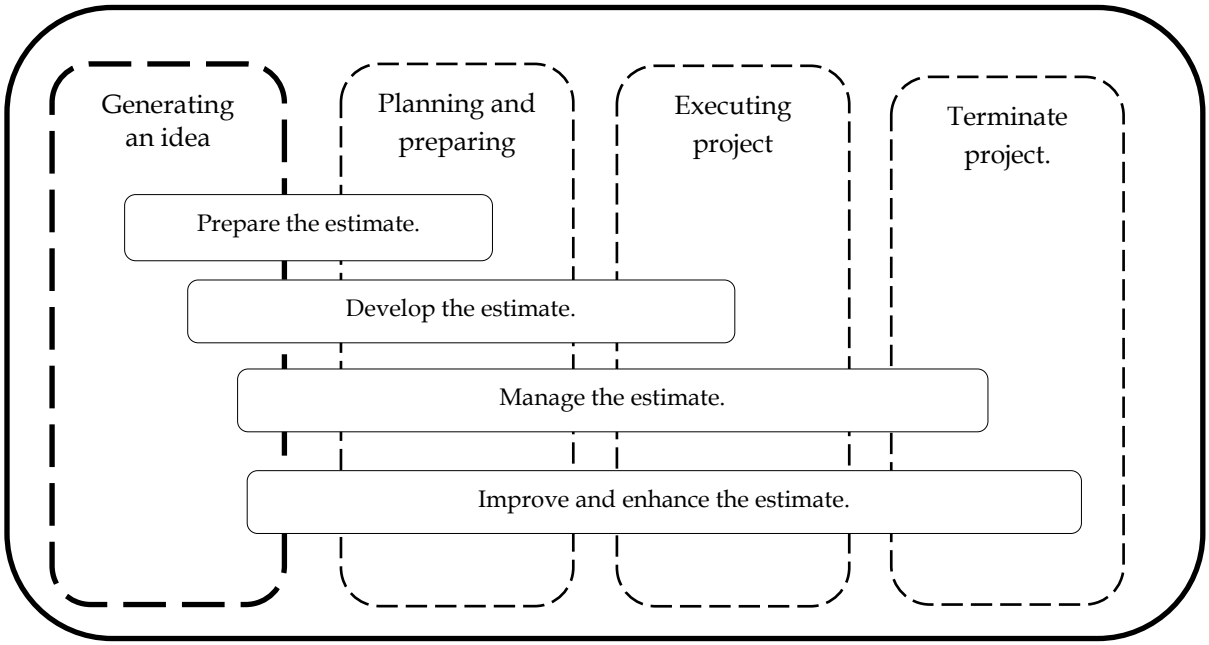

Figure 3: Enhancing estimation methods in software development process.

8.2 Limitation of study

One limitation of the study is that the time schedule for the software development is small (just eight weeks) and does not give a general overview of the differences between the results calculated for the estimation methods. A bigger project, scheduled to be completed over a long period of time (maybe over a year) with more developers involved, could give a better view if the bottom-up estimate is better than the top-down estimate or vice-versa.

Comparing the baseline estimate with just two estimation methods (i.e., bottom-up and top-down estimation method) did not give the possibility of comparing other methods of estimation like the algorithm methods that could have used different metrics and equations to derive the total effort required to complete the project.

8.3 Future research opportunities

More research has been conducted to develop better estimation models, reduce the time spent creating the estimate or find a way to do things differently. All these have made researchers to come up with the concept of "No Estimate model".

The No Estimate model is not proposed to eliminate the idea of estimating, but to investigate another method of solving the problems of delivering software projects on time. The steps suggested are:

- Risk Estimation Method: This is process of arranging the work structure of a software project to get as much info as possible and get started on the bigger risk first. The aim of this method is that it is easier to see if the project will be completed or not.
- Percent Complete method: This is the process whereby a small part of the project selected and worked on over a chosen time frame (like 2 weeks or a month). This will give the stakeholders the opportunity to gain better understanding of how long the project could take and if it is worth it. After the 2 weeks, if things look bad, the stakeholders can stop investing into it. If things work fines, the next major decision point like a month or 3 months can be chosen and the whole process repeated.

REFERENCES

Abrahamsson, S. & Ronkainen, W. (2002). *Agile software development methods: Review and Analysis*. Espoo, Finland, Technical Research Centre of Finland, VTT Publications 478, available online: http://inf.vtt.fi/pdf/publications/2002/P478.pdf.

Adams, R. & Schvaneveldt, J. (2003). *Understanding Research Methods.* 4th Edition, Pyrczak Publishing.

Albert L, Lederer and Jayesh, P. (1992). *Nine Management Guidelines for Better Cost Estimation.* CACM, Vol 35

Albrecht, A.J (1979). *Measuring Development Productivity*. IBM California. 83 – 92

Beck (2000). *Extreme programming explained*. Reading, MA, Addison Wesley Longman Inc.

Beck, Fowler (2000). *Planning extreme programming*. New York, Addison-Wesley.

Bill, S. (2020). *Importance of Project Schedule and Cost Control in Project Management*. Global Knowledge. https://www.globalknowledge.com/ca-en/resources/resource-library/articles/importance-of-project-schedule-and-cost-control-in-project-management/

Boehm, B. (1984). *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J.

Boehm, et al. (2000). *Software Cost Estimation with COCOMO II*. Prentice Hall, Upper Saddle River, New Jersey USA.

Boehm, B. (2000). *Safe and Simple Software Cost Analysis*. IEEE Software, Prentice Hall, Upper Saddle River, New Jersey USA.

Boehm, B. (2001). *Software Engineering Economies. In Pioneers and their contributions to Software Engineering,* Prentice Hall, Upper Saddle River, New Jersey USA.

Boehm, B. (2010). *Current and Future Challenges for Software Cost Estimation and Data Collection,* Prentice Hall, Upper Saddle River, New Jersey USA.

Brooks, F. (2003). *Three Great Challenges for Half-Century-Old Computer Science*. Journals of the ACM. Vol 50, Pg. 25-26.

Campbell, L. and Brian, K. (1995). *Software Metrics: Adding Engineering Rigor to a Currently Ephemeral Process*, briefing presented to the McGrummwell F/A-24 CDR course.

Capers, J. (2008). *Preventing Software Failure: Problems Noted in Breach of Contract Litigation*, Caper Jones & Associates, Narragansett, RI.

Capers, J. (2007). *Estimating Software Costs: Bringing Realism to Estimating*. 2nd Edition.

Cavalieri, S. Maccarrone, P. & Pinto, R. (2004). *Parametric vs Neural Network Models for the Estimation of Production Costs: A Case Study in Automotive Industry.* International Journal of Production Economics.

Charette, R. (2005). *Why software fails: We waste billions of dollars each year on entirely preventable mistakes*. IEEE Spectrum Spotsylvania, Va.

Conte, S. Dunsmore, H. & Shen, V. (1986). *Software Engineering Metrics and Models*, Menlo Park, Ca.

Dehaghani, S & Hajrahim, N. (2013). *Which Factors Affect Software Projects Maintenance Cost More?*

Eberendu, A. (2014). *Software Project Cost Estimation: Issues, Problems and Possible Solutions*, Madonna University, Nigeria

Enachescu, C. & Dumitru, R. (2009). *Software Cost Estimation Model based on Neural Networks*.

Garmus, D. & Herron, D. (1995). *Measuring the Software Process: A Practical Guide to Functional Measurement. Prentice* – Hall, Englewood Cliffs.

Garmus, D & Herron, D. (2001). *Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, Boston, Mass.

Henrich, A. (1997). *Repository Based Software Cost Estimation*

Hihn, J. & Habib-agahi, H. (2000). *Reducing Flight Software Development Cost Risk: Analysis and Recommendations*, Long Beach, CA, 2000.

Hihn, J. & Habib-agahi, H. (2000). *Cost Estimation of Software Intensive Projects: A Survey of Current Practices*. http://www.ispa-cost.org/ - International Society of Parametric Analysts.

Hull, D. (2009). *Methods and Challenges in Early Cost Estimation*

Humphrey, W. (1989). *Managing the Software Process*, Addison Wesley, Don Mills Ont.

Humphrey, W. (1995). *A discipline for software engineering*. Addison Wesley.

Hung, R. (2006). *Business Process Management as Competitive Advantage: A Review and Empirical Study*. Total Quality Management & Business Excellence

Iqbal, S. Idrees, M. Sana, A. &Khan, N. (2017). *Comparative Analysis of Common Software Cost Estimation Modelling Techniques. Mathematical Modelling and Applications*. Vol. 2, No. 3, pp. 33-39

Isabella, W. (2002). *Improved Software Cost Estimation: A Robust and Interpretable Modelling Method and a Comprehensive Empirical Investigation*

Jeffries, Anderson, Hendrickson (2001). *Extreme Programming Installed*. Upper Saddle River, NJ, Addison-Wesley.

Jones, C. (2005). *Software Quality in 2002: A Survey of the State of the Art. Software Productivity Research*, Marlborough, Massachusetts.

Jones, C. (2005). *Software Cost Estimating Methods for Large Projects: Software Productivity Research*, LLC.

Jones, T. (1998). *Estimating Software Costs*, New York: McGraw-Hill.

Karen, L. Bramble, M. & Hihn, J. (2003). *Handbook for Software Cost Estimation.*

Kansala, K. (1997). *Integrating risk assessment with Cost Estimation*. Nokia Research Center Espoo. Vol 14, Issue 3, 61-67 Software IEEE.

Kemerer, F. (1987). *An Empirical Validation of Software Cost Estimation Models*

Khatibi, V. & Jawaw, D. (2011). *Software Cost Estimation Method: A review Journal of Emerging Trends in Computing and Information Science*.

Koskinen J. (2010). *Software Maintenance Costs*. Jyväskylä: University of Jyväskylä.

Kruchten, P. (2007). *Software Development Cost Estimation*. University of British Columbia.

Laudon, K. & Laudon, J. (1999). *Essentials of Management Information Systems*. 3rd International Edition. Prentice Hall. 27-30

Leung, H and Zhang, F. (2001). *Software Cost Estimation. Department of Computing*, The Hong Kong Polytechnic University.

Liming, W. (1997). *The Comparison of the Software Cost Estimating Methods*

Linda, M. (2006). *The Limitations of Estimation for IT Professionals*. Volume 8.

Malevanny, S. (2005). *Software Project Cost Estimates Using COCOMO 11 Model Introduction to Software Cost Estimation*. http://www.mhprofessional.com/downloads/products/0071483004/0071483004_ch01.pdf

McConnell, S. (1998). *Software Project Survival Guide*. Microsoft Press

McConnell, S. (2006). *Software Estimation: Demystifying the black art*, Microsoft Press, Redmond WA.

Miles, M. & Huberman, A. (1994). *Qualitative Data Analysis: An Expanded Sourcebook*. 2nd Edition, Thousand Oaks, Sage, California.

Mittal, H & Bhatia, P. (2002). *A comparative study of Conventional Effort Estimation and Fuzzy effort estimation based on Triangular Fuzzy Numbers.*

Moody, D. (2002). *Empirical Research Methods*. Available online at http://www.idi.ntnu.no/ekaterip/dif8916/Empirical%20Research%20Methods%20Outline.pdf.

Nahar, N. (2001). *Information Technology Supported Technology Transfer Process. A Multi-site Case Study of High-tech Enterprises*. Jyväskylä Studies in Computing.

Nguyen, V. Deeds-Rubin, S. Tan, T. & Boehm, B. (2007), *A SLOC Counting Standard*, Center for Systems and Software Engineering, University of Southern California.

Nguyen Vu. (2010) *Improved Size and Effort Estimation Models for Software Maintenance*. University of Southern California.

Novack, R. & Simco, S. (1991). *The Industrial Procurement Process: A Supply Chain Perspective.* Journal of Business Logistics.

Park, R. (1995). *A Manager's Checklist for Validating Software Cost and Schedule Estimates*.

Prasad, B. & Harker, P. (2009). *Examining the contribution of IT towards Productivity and Profitability in the US retail Banking.*

Putnam, L. (2003). *Five core metrics: the intelligence behind successful software management.* Dorset House Publishing. ISBN 0-932633-55-2

Putnam, L. (1978). *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-4,

Ravidranath, C. (2004). *Software Metric: A Guide to Planning Analysis and Application*

Roger, S. (1992). *Software Engineering: A Practitioner's Approach*, 3rd Edition, McGraw-Hill, Inc., New York.

Rosalind, L. Pfleeger, L. & Wu, F. (2005). *Software Cost Estimation and Sizing Methods: Issues and Guidelines*

Roy, R. (2003). *Cost Engineering: Why, What and How*. Available on at http://www.cranfield.ac.uk

Shepperd, M. (1996). *Effort Estimation Using Analogy*. IEEE

Stamelos, I. Angelis, L. Morisio, M. Sakellaris, E. & Bleris, G. (2003). *Estimating the development cost of custom software. Information & Management.*

Stevenson, C. (1995). *Software Engineering Productivity*, London.

Strike, K. & Emam, K. (2001). *Software Cost Estimation with Incomplete Data*. IEEE, Vol 27, Pg. 890-908

Stutzke, R. (2005). *Estimating Software – Intensive Systems: Projects, Products and Processes.* Addison-Wesley, Boston, Mass.

Touesnard, B. (2004). *Software Cost Estimation: SLOC-based Models and the Function Point Model Version 1.1*

Zawrotny, S. (1995). *Estimating Software Development Projects: No Silver Bullets, But Vendor Software Does Help?*

Zia, Z. Rashid, A. & Zaman, K. (2011). *Software cost estimation for component based fourth generation-language software applications*, Software, IET, vol.5, no.1, pp.103-110.

Appendix A - List of abbreviations and terms used in this study.

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 one of the methods 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 also one of the methods used to estimate the effort required for the software development.

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 based on 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.

Software Cost Estimation Process

Software cost estimation process is the set of techniques and procedures that an organization uses to arrive at a software cost estimate. Generally, there is a set of inputs to the process (e.g., system requirements) and an output of effort, manpower loading, and/or duration. It is discovered that this process at an early stage of software development could be very difficult task to achieve.

Work Breakdown Structure

A Work breakdown structure (WBS) is a deliverable oriented decomposition of a project into smaller components. It defines and arranges a project's discrete work elements in a way that helps organize and define the total work scope of the project which provides the necessary framework for detailed cost estimating and control along with providing guidance for schedule development and control.

Request for Proposal

A request for proposal (RFP) is a document that an organization posts to elicit bids from potential vendors for a product or service. The quality of an RFP is very important to successful project manage because it outlines the bidding process and contract terms and provides guidance on how the bid should be formatted and presented. A RFP is typically open to a wide range of bidders, creating open competition between companies looking for work.

Appendix B - Some useful datasets.

Table 14: Detailed summary of distribution of effort during development

Id	Collected data	Release 1	Release 2	Release 3	Release 4	Release 5	Correction Release	Total
1	Calendar time (weeks)	2	2	2	1	1	0.4	8.4
2	Total work effort (h)	195	190	192	111	96	36	820
3	Task allocated actual hours	136(70%)	95(50%)	118(61%)	51(46%)	42(44%)	27(75%)	469(57%)
4	# LOCs implemented in a release	1821	2386	1962	460	842	227	7698
5	Team productivity (loc/hr)	13.39	25.12	16.63	9.02	20.05	8.4	16.90
6	Code integrations (integrations/day)	8.1	10.1	7.9	10.5	8.2	8.5	8.9
7	Avg. time between integration (minutes)	26	21	40	31	27	30	29
8	Avg. number of files per integration	1.7	2.4	3.1	2.6	3.0	3.0	2.6
9	# User stories implemented	5	9	9	4	3	4	34
10	# User stories postponed for next release	0	1	0	1	2	0	4
11	User story effort (actual, median, h)	10.1	8.3	7.6	5.9	5.2	2.8	6.8
12	User story effort (actual, max, h)	63.1	26.9	41.7	21.8	15.9	7.6	63.1
13	# Tasks defined	10	30	18	21	19	9	107
14	Task effort (actual, median, h)	11.7	2.9	5.9	1.7	2.6	0.7	2.7
15	Task effort (actual, max, h)	32.3	8.8	14.0	8.8	5.3	3.4	32.3

16	# post-release defects	4	5	4	4	11	-	2.19
17	Post-release defects/KLoc	2.19	2.10	2.04	8.7	13.06	-	1.43 (3.75)
18	# Post-release enhancement suggestions made by testers	17	13	5	3	0	-	38
19	Pair programming (%)	81.7	76.3	73.0	78.8	54.2	90.4	75.9
20	Required customer involvement (%)	17.4	21.4	18.6	25.0	23.4	24.3	20.6
21	Rework costs (%)	-	8.7*	11.8	11.6	2.6	61.5	9.8

*Includes also enhancements.

Table 15: Defects

	Defects							
#	Date found	During Task	Description	Severity	Release In	Release Out	Date fixed	Time fixed
1	2003-02-14	Pre-1 Test	User can enter html-directives into text fields which can in worst case enables script writing onto page.	Major		R1	2003-02-14	60
2	2003-02-14	Pre-1 Test	Create forms leave user to "ack" page, should open the created folder/workspace/folder in which file was created.	Cosmetic		R1	2003-02-14	30
3	2003-02-14	Pre-1 Test	Refreshing form left by defect #2 causes action to be performed again, ie creating another folder or file with same attributes.	Major		R1	2003-02-14	60
4	2003-02-14	Pre-1 Test	File link requirements are too strict? (3.1)	Cosmetic		R1	2003-02-14	60
5	2003-02-14	Pre-1 Test	after newFileHandeler file description losses spaces	Cosmetic		R1		
		8.1.	New File -function created duplicated file (same file x2) when performed, action didn't repeat itself (ghost?)	Minor?	R2			
	2003-02-19	9.1.	top-level description converts wrong	Minor?	R2			
	2003-02-20	10.1.	DateCreateded was update, becourse	Minor?	R2		2003-02-20	30

			dateCreatede type was timestamp					

Table 16: Change History

Version	Date	Comments
0.1	03.02.2003	first draft, for R1
0.2	10.02.2003	fixed findings made by Abr
1.0	11.02.2003	fixed findings found in steering group meeting
1.1	18.02.2003	Update schedules and estimates for R2
1.2	03.03.2003	Updates schedules and estimates for R3
1.3	04.03.2003	fixed estimates
1.4	06.03.2003	fixed spent time, concerning task miscellaneous
1.5	17.03.2003	updated actual hours for R3, added schedule and estimated for R4
1.6	24.03.2003	updated actual hours for R4
1.7	27.03.2003	updated schedule and estimates for R5
1.8	28.03.2003	updated actual hours for R5 and created "post R5 release"
1.9	03.04.2003	updated estimates and post-R5 to R6
2.0	16.04.2003	Final version after last post-mortem