COMPARATIVE ANALYSIS BETWEEN FPA AND COCOMO TECHNIQUES FOR SOFTWARE COST ESTIMATION

ABU BAKER ALI MOFTAH

A thesis submitted in
fulfillment of the requirement for the award of the
Degree of Master of Computer Science (Software Engineering)

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

JULY 2014

ABSTRACT

Software cost estimation is the process of predicting the effort required to develop a software system. The basic inputs for the software cost estimation are programs, size and set of cost drivers, while the output is effort in the form of person-month and cost. In this thesis, Function Point Analysis (FPA) and Constructive Cost Model (COCOMO) have been used to estimate software project cost of two case studies. They are Web-Based Dog's Diseases Diagnosis System (WBDDDS) and Sugar Bun Online Bakery System (SBOBSE). By using FPA, it was shown that for the WBDDDS, the person-month was 12.506 with the total cost of USD65,031.2 were estimated. While using COCOMO, it was shown that 16.286 persons-month with the total cost of USD 84,687.2 were estimated. However, for the SBDBSE, by using FPA, 19.62 persons-month with the total cost of USD102,024 were estimated. It also shown that 19.354 persons-month with the total cost of USD100,640.8 were estimated by using COCOMO. In conclusion, there are no best techniques to estimate cost for a project. It all depends on the parameters of a system.

ABSTRAK

Membuat anggaran kos perisian adalah proses meramalkan usaha yang diperlukan untuk membangunkan sesebuah sistem perisian. Input asas untuk membuat anggaran kos perisian adalah program, saiz dan set pemacu kos, manakala untuk output usaha adalah dalam bentuk bilangan orang diperlukan bagi tempoh sebulan iaitu *person-month* dan kos. Dalam kajian ini, *Function Point Analysis (FPA)* dan *Constructive Cost Model (COCOMO)* telah digunakan untuk menganggarkan kos projek perisian untuk dua kajian kes. Kajian kes tersebut adalah *Web-Based Dog's Diseases Diagnosis System (WBDDDS)* dan *Sugar Bun Online Bakery System (SBOBSE)*. Dengan menggunakan FPA, hasil anggaran kos ke atas WBDDDS menunjukkan bahawa sebanyak 12.506 *person-month* dan jumlah kos sebanyak USD65.031,2 diperlukan. Manakala, dengan menggunakan COCOMO, 16.286 *person-month* dengan jumlah kos sebanyak USD4,687.2 dianggarkan. Walau bagaimanapun, bagi SBOBSE, dengan menggunakan FPA, 19.62 *person-month* dengan jumlah kos sebanyak USD102.024 dianggarkan. Selain itu, kajian juga menunjukkan 19,354 *person-month* dengan jumlah kos sebanyak USD 100,640.8 dianggarkan dengan menggunakan COCOMO. Kesimpulannya, tiada satu teknik terbaik untuk membuat anggaran kos. Anggaran kos yang baik adalah bergantung kepada parameter sesebuah sistem.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

LOC	-	Line of Code
SLOC	-	Source Line of Code
FPA	-	Function Point Analysis
COCOMO	-	Constructive Cost Model
EAE	-	Effort Adjustment Factor
UPA	-	Unadjusted Function Point

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

Estimating software development cost remains a complex problem, one that continues to attract a considerable amount of research attention. Improving the accuracy of the cost estimation models available to project managers would facilitate a more effective control of time and budgets during the software development. The needs for a reliable and accurate cost estimation in software engineering have been an ongoing challenge for software engineers in the last decade [1] [2] [3].

The Standish Group Chaos Report recently reported that about 66% software projects are delivered with some delay, over-budget, and many are not even finished. Commonly, the main cause of these problems is the failure of the software development cost estimation (SDCE) [4].

The software cost estimation is the process of predicting cost for the development of the software. The software cost is the amount of cost in either person days or person hours necessary for conducting the tests. The most commonly used methods for predicting software development cost are Function Point Analysis, Constructive Systems Engineering Cost Model (COSYSMO), SEER for Software (SEER-SEM), Putnam model, and Constructive Cost Model (COCOMO) [5].

The function point analysis (FPA) is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user. The function does not depend on the programming languages or tools used to develop a software project. FPA is a standard method to measure the software development from the user's point of view. The past three decades of the use of FPA have shown that it is a proven method [6] [7] [8].

The Constructive Cost Model (COCOMO) is developed by Boehm. It is based on the linear-least-squares regression. Using the line of code (LOC) as the unit of measure for the software size itself contains so many problems. These methods failed to deal with the implicit non-linearity and interactions between the characteristics of the project and effort [2]. This research looks into the both techniques and compares them in term of cost involved.

1.2 Problem Statement

Software cost estimation is the process related to the well-balanced management of a software project. The most commonly used methods for predicting software costs estimation are function point analysis (FPA) and Constructive Cost Model (COCOMO) [6] [9]. Despite the evolving research activity, the task of estimating accurately the budget and the delivering time has been a research problem for many decades. Nowadays, the cost of a project is still estimated with error. Therefore in this study, the use of Function Point Analysis (FPA) and Constructive Cost Model (COCOMO) is compared. The techniques are used to compare the software cost estimation for the two case studies to get the person-month and total cost.

1.3 Project Objectives

The main objectives of this study are:

(i) To evaluate the estimated software cost using the function point analysis (FPA) estimation technique in the two case studies of the Web-Based Dog's Diseases Diagnosis System (WBDDDS) and the Sugar Bun Online Bakery System, E-Sugarbun (SBOBSE).

(ii) To evaluate the estimated software cost using the Constructive Cost Model (COCOMO) technique in the two case studies of the Web-Based Dog's Diseases Diagnosis System (WBDDDS) and the Sugar Bun Online Bakery System, E-Sugarbun (SBOBSE).

(iii) To compare the software cost estimation using FPA and COCOMO in terms of time of person-month and total cost for both case studies: Web Based Dog’s Diseases Diagnosis System (WBDDDS) and Sugar Bun Online Bakery System, E-Sugarbun (SBOBSE).

1.4 Scope of the Project

This study focuses on the comparison of the software cost estimation using FPA with External Input, External Outputs, External Inquiry, Internal Logical File and External Interface File as the parameters. While for the Constructive Cost Model (COCOMO) parameters, Basic COCOMO and Intermediate COCOMO were used. The LOC for WBDDDS and SBOBSE are calculated manually and applied equations as provided by FPA and COCOMO.

1.5 Thesis outline

The thesis consists of five chapters. Chapter 1 is where the overview, main objectives and scope of works of the project were carried out. Chapter 2 illustrates the related literature review of this project. Chapter 3 discusses the methodology to obtain the entire objectives of this project. Chapter 4 explains the implementation and the detailed steps in this work. Finally, Chapter 5 includes the objectives achievement, disadvantages, future work, and conclusion of this project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The software cost estimation subject has been a dynamic exploration range, with the examination expanding considerably in the course of the last few decades. Reviews written in [10] and [11] indicate that the exploration in the recent 25 years concentrated on diverse levels of software estimation.

Constructive Cost Model (COCOMO) is a software model, created by Barry Boehm, which focuses around algorithms for the estimation of costs. A fundamental relapse recipe is used with the parameters acquired from the task data of the undertaking qualities from the past, present, and future [12].

According to Pressman [13], he indicated that the Function Point Analysis is a well-known method to estimate the size of the software systems and software projects, so the function point count can be applied to development projects. There are 5 significant segments of the Function Point Analysis, which catch the practicality of the provision including the external Inputs (EIs), external Outputs (EOs), external Inquiries (EQs), internal Logical Files (ILFs) and external Interface Files (EIFs).

2.2 Overview of Cost Estimation

Pandian [14] suggested the Analogy, Top down and Bottom up approaches as the three estimation methodologies in which the Analogy method estimates the project by using the historical data of the previously completed projects and comparing with the already existing information on the completed projects. The second approach concentrates on the overall characteristics instead of the functional and non-functional requirements of the system to be developed, whereas the bottom up approach considers each and every component and then combines them all to give the overall required estimation for the project, which is found to provide the most detailed estimation.

McConnell [15] reported that numerous projects were either cancelled or missed its delivery dates. However, more than half of the projects substantially overrun their estimation, as shown in Table 2.1, from which this approach was based on the several surveys conducted. The related studies indicated that the effective software estimation is one of the most important and difficult software development activities [16]. The over-estimating system and the under-estimating system of a project are both bad for different reasons, which the overestimating will cause a project to take at least as long as it was estimated. However for the other system (under-estimating), a project will lead to under staffing, under scoping the quality assurance effort, and short schedule [17].

Table 2.1 Software Overrun Case Studies [16]

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

The previous studies indicated several reasons for the overruns of the cost estimation as which were listed by Laird [18], who found that they are including of the lack of training, education, confusion of the desired schedule/effort target with the estimate, and creeping requirements affected the software cost estimation. On the other hand, the researchers identified other reasons to exceed the requirements of the project, which is incomplete, unclear and difficult in managing the project schedule, such as changing the scope, planning to schedule more assertive than necessary, and insufficient resources for the project.

In view of the Khatibi, Jawawi and Dayang [19] research, which found the reasons for the failure of the software projects during their intensive research on the internet sites, which showed that poor planning of the project, insufficient requirements engineering, suddenly decisions at the early stages of the project and inaccurate estimations that are considered as the most important reasons.

In another study conducted by Boehm [20] who is known as the leader of the software cost estimation from which he also reformulated his model in COCOMO II in 1997, that consists of three different sub models: application composition, early design, and post-architecture. The researcher suggested three basic reasons for failure of cost estimations, including the lack of clear understanding of the software requirements, under-estimation of the software size, and the required effort for the software projects [16].

Boehm [20] commented that there are large numbers of cost analysis methods available, but found that these are not always safe to be used. The simplest method is to base cost estimate on the typical costs or productivity rates of the previous projects. Some of the simple methods are useful if the new project does not have any cost-critical differences from the previous projects. However, they are risky if the critical factor of the cost driver has been discarded.

Software cost estimation is an important, but difficult. In the last there decades, different models based on techniques were proposed, such as SLIM, Checkpoint, Price-S, SEER-SEIM, ESTIMACS, and COCOMO. When most of the researchers were working on developing the cost estimation, they found the same difficulties once the software grows in size and complexity, which makes it very difficult to predict the cost of software development [21]. Whereas three models were created that are significantly used for cost estimation, which are known as Boehm's COCOMO, Putman's SLIM, and Albrecht's function point. Most of the models used the size measurement methods, such as Line of Code (LOC) and Function Point (FP) for determining the cost estimation. The accuracy of the cost estimation is directly related with the estimation of size [16]. In this research however, the COCOMO and FPA were used to evaluate estimated software cost for two case studies.

2.3 Cost Estimation Techniques

This method is formed to give a mathematical approach to carry out the software estimates. These mathematical equations are based on the research and historical data and used inputs, such as Source Lines of Code (SLOC), and some other cost drivers; these algorithmic models have been extensively worked on. Several models have been formed based on these, such as the COCOMO models, function point and Putnam models that are also known as based models [22]. There are many ways in the literature to estimate the cost. Basically, the cost estimation methods are classified into two groups, which are arithmetic and non-arithmetic [23]. In this study, the arithmetic method will be used to discuss the estimate of the cost.

Constructive Cost Model commonly referred to as COCOMO, which is actually a hierarchy of three models of an increasing detail, is based on a study of sixty-three projects developed at TRW from the period of 1964 to 1979. In his text, Boehm describes the development of COCOMO as being the result of a review of then available cost models coupled with a Delphi exercise that resulted in the original model. This model was calibrated using a database of 12 completed projects [24].

2.3.1 Constructive Cost Model (COCOMO)

The Basic COCOMO registers advancement exertions and cost as a system capacity communicated in lines of code (LOC). The essential steps that are included in this model are to get a starting assessment of advancement from the 1000s of evaluated conveyance lines of source codes. Also to decide on a set of 15 various components from the distinctive traits of the undertaking, and to settle the exertion gauge by duplicating the introductory appraisal with the elements.

The starting evaluation, which is additionally alluded to as the ostensible appraisal, is dictated by the static single variable model comparison utilizing Kilo Lines of Code (KLOC) as the measure of size; this decides the starting exertion in an individual month, where the Development Mode in this research is Semi Detached ($3.0*(KLOC)^{1.12}$) that relies on the kind of the undertaking, as demonstrated in Table 2.2 and emulating development mode [22].

$$EFFORT = a * (KLOC)^{b} \qquad (2.1)$$

Table 2.2 Basic COCOMO Calculating Person Months [22]

Development Mode	Basic Effort Equation	Time Duration
Organic	$\text{Effort} = 2.4\ \text{KLOC}^{1.05}$	$\text{TDEV} = 2.50 *(\text{PM})^{0.38}$
Semi Detached	$\text{Effort} = 3.0\ \text{KLOC}^{1.12}$ (2.1)	$\text{TDEV} = 2.50 *(\text{PM})^{0.35}$
Embedded	$\text{Effort} = 3.6\ \text{KLOC}^{1.20}$	$\text{TDEV} = 2.50 *(\text{PM})^{0.32}$

2.3.1.1 Intermediate COCOMO

This model processes advancement exertion of the software as a system size capacity and a set of cost drivers, these incorporate subjective evaluations of the fittings, work force and undertaking traits, and items. The cost drivers can be put into groups, as shown in Table 2.3.

Table 2.3 Categories of Intermediate COCOMO [22]

Cost Drivers			
Product attributes	Hardware attributes	Personnel attributes	Project attributes
Size of application database	Memory Constraints	Software Engineer Capability	Application of software engineeri ng methods
Complexity of the product	Volatility of the virtual machine environment	Analyst Capability	Use of software tools
Required Software Reliability	Run-time Performance Constraints	Virtual Machine Experience	Required development schedule
	Required Turnaround Time	Applications Experience	Application of software engineeri ng methods
		Programming Language Experience	Use of software tools

It extends from a high to a low in the matters of worth. An exertion multiplier, focused around the evaluations, is chosen from the tables that was distributed by Boehm, and an exertion appraisal component (EAF) is received as an item from these multipliers. The typical values for EAF range from 0.9 to 1.4 [30]. The intermediate COCOMO model takes the following form:

$$\text{EFFORT} = a^{*} (\text{KLOC})^{b} {*}\ \text{EAF} \qquad (2.2)$$

Where the effort is applied in person-months, KLOC is the estimated number of thousands of delivered lines of code for the project, and EAF is the factor calculated. The coefficient “a” and the exponent “b” use semi detaches mode value, as given in Table 2.4.

Table 2.4 Intermediate COCOMO Calculating Person-Months [22]

Development Mode	Intermediate Effort Equation
Organic	Effort = 3.2 * (KLOC) $^{1.05}$* EAF
Semi Detached	Effort = 3.0 * (KLOC) $^{1.12}$ * EAF (2.2)
Embedded	Effort = 2.8 * (KLOC) $^{1.20}$ * EAF

The same basic equation for the model is used, but fifteen cost drivers are rated on a scale from 'very low' to ‘very high’ to calculate the specific effort multiplier and each of them returns an adjustment factor, which multiplied yields in the total EAF (Effort Adjustment Factor). The adjustment factor is 1 for a cost driver that's judged as normal. In addition to the EAF, the model parameter "a" is slightly different in Intermediate COCOMO from the basic model. The parameter "b" remains the same in both models [22]. For example, if modern programming practices are used, the initial estimates are scaled downward by multiplication with a cost driver having a value less than 1. If there are stringent reliability requirements on the software product, this initial estimate is scaled upward. Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then depending on these ratings, he suggests appropriate cost driver value that should be multiplied with the initial estimate, which is obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes.

2.3.2 Function Point Analysis (FPA)

The FPA is another method used to quantify the size and complexity of the software system on the functions that the system provides its user. A lot of exclusive models regarding cost estimates have a function pointer approach, such as ESTIMACS and SPQR/20 [15]. This measurement is based on the program’s functionality introduced by Albrecht [25]. The number of distinct types decides the total number of function points. Mainly, the two steps are followed in the function points counting the User Functions: the real count of function points is achieved by keeping in mind a linear arrangement of five basic software component basics, such as External Inputs, External Outputs, External Inquiries, Logic Internal Files, and External Interfaces [26].

The above are all at the complexity level out the following three levels: simple, average or complex. The total of these numbers based on the complexity level is the number of function counts (FC). The Environment Processing Complexity-based Adjustment is the last function point that is obtained by the multiplication of FC with an adjustment factor that is decided by contemplating 14 processing complexity aspects. The FC can be modified to a maximum of 35% or -35% with the help of the adjustment factor [27].

The function point analysis is a gauge for sizing and is associated with a clear business significance. It was first made public by Allan Albrecht of IBM in 1979 and is designed to measure commercial type applications. It is not suitable for applications, such as technical or scientific. These applications are more complex than the method of feature points that is not designed to handle algorithms. The approach of function points has characteristics that overcome the major problems encountered when using lines of code as a measure of the system size [28].

The FPA is system for evaluating the span of activities of software frameworks and software. Initially, the system was utilized within the early phases of the waterfall model so that the exertion of execution could be evaluated and focused around the conduct of data and yield, as characterized in the utilitarian documentation, the size and unpredictability of software expansions, it gets to be progressively significant to create powerful cost of a fantastic software inside a specified period [29].

The software size helps in developing an initial estimate for the software effort/cost estimation during the software development life cycle. The COCOMO model provided this estimate based on the source lines of code (SLOC). It was reported that SLOC produced many problems [21]. For example, in the modern software programming, auto-generate tools produced a large number of LOC. SLOC also changes with the developer's experience, difference in programming languages, variation in the graphical user interface (GUI) code generation, and lack of functionality, The estimation of SLOC under this condition seems uncertain to measure, which is why Albrecht proposed his idea of computing the software size based on the system functionality [30].

In 1979, Albrecht [25] published his article on the FP methodology while he was working at IBM. He proposed that FP has no dimension and that FP was computed based on the analysis of project requirements. The requirements helped in identifying the number of function to be developed along with the complexity of each function. Once the number of FP is measured, the average number of function points per month was specified, and the labour cost per month is estimated; the total budget can be computed. Albrecht originally proposed four function types, which are files, inputs, outputs, and inquiries with one set of associated weights and 10 General System Characteristics (GSC). In 1983, the work, developed by Albrecht and Gaffney, proposed the expansion of the function type, a set of three weighting values (i.e. simple, average, and complex) and fourteen General System Characteristics (GSCs).

Kemerer [31] provided a famous study reporting the results of the comparative accuracy for four software cost estimation models. They are the Function Points, SLIM, COCOMO, and ESTIMACS. The results were produced using the data collected from 15 completed software projects. Each model was tested based on its predictive capability on the computing software cost. The results showed that the models require substantial calibration. The researcher also identified the main attributes that affect the software's productivity. Recently by using Albrecht's FPA method and using an analogous approach, the authors provided a methodology that they claimed as more reliable and accurate in predicting the software size at an early stage of the software life cycle. Recently, FP gains more attention as a powerful approach for estimating software effort [32].

There are two parts in the model that are Unadjusted Function Point (UFP) and Adjusted Function Point (AFP). The UFP consists of five components. They are External Inputs (EI), External Outputs (EO), External Inquires (EQ), Internal Logical Files (ILF), and External Interface date (EIF) [32].

Table 2.5 Function Types and Weights [32]

Function Type	Simple	Average	Complex
External Input	3	4	6
External Output	4	5	7
Internal Files	7	10	15
External Files	5	7	10
External Inquiry	3	4	6

There are 14 GSCs components that affect the length of the project energy and each can be ranked from no influence to necessary (0-5). They were connected with 14 factors called f1, f2... f14. These types of factors are outlined in Table 2.5. The sum of the Table 2. 5 components are then multiplied with the given Equation 2.3, which constitute this Adjustment Factor (AF) defined within the range. (0.65, -1.35) [28]. For example, the value adjustment factor (VAF) is based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted. Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence. The International Function Point Users Group (IFPUG) Counting Practices Manual provides detailed evaluation criteria for each of the GSC'S. Table 2.6 is intended to provide an overview of each GSC.

$$\text{AF} = 0.65 + 0.01 \sum_{i=1}^{14} \text{fi} \qquad (2.3)$$

Table 2.6 General System Characteristics (GSCS)[28]

GSCS's Factors	Rank					
	0	1	2	3	4	5
Data Communications	No Influence	Incidental	Moderate	Average	Significant	Essential
Distributed Functions						
Performance						
Heavily Used Configuration						
Transaction Rate						
Online Data Entry						
End User Efficiency						
Online Update						
Complex Processing						
Reusability						
Installation Ease						
Operational Ease						
Multiple Sites						
Facilitate Change						

Then, the Unadjusted FP will be then multiplied by the AF to develop the AFP count, as given in Equation 2.4. The AFP value is definitely within 35% in the original UFP physique. A diagram, which shows the method of computing FP, is given in Figure 2.1 [32].

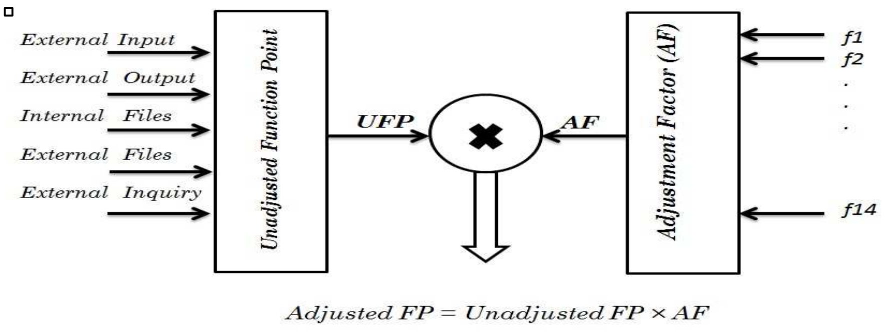

Figure 2.1 Function Point Computation Model [32]

$$\text{Adjusted FP} = \text{Unadjusted FP} \times \text{AF} \qquad (2.4)$$

The Adjusted Function Point is easily determined using the Equation 2.4. As mentioned in the previous section, the AF can vary from 0.65 to 1.35, so the AF exerts an influence, the final Adjusted FP.

2.4 Related Work

Kemerer [31] evaluates four software cost estimation models, which are SLIM, COCOMO, Function Point and ESTIMACS. This research had used data on 15 large projects. He found that the models do not suitable for business data processing environment. Heemstra and Kusters [33] do an experiment on the effectiveness of FPA model. Their research had used data of Dutch organizations. They found that FPA is more acceptable for sizing measurement. While, Sheta and Aljahdali [32] had done some enhancement on COCOMO and FPA using fuzzy model. They found that the proposed fuzzy model show better estimation. However, this research limited to compare on software cost estimated between COCOMO and FPA.

2.5 Summary

This specific chapter reviewed the actual FPA and COCOMO type. It also gives a brief explanation concerning other estimations. The following chapter will look into research methodology on the study.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter discusses the methodology that was used for this research. This research was conducted by applying the five-phase estimation using FPA in the case studies. It was also conducted by applying the software cost estimation (COCOMO) in the case studies and by comparing the software cost estimation using FPA and COCOMO for these two case studies.

3.2 Research Activities

This section shows steps for the two techniques to be applied for these two case studies. The first technique is COCOMO and it comes with the following steps: Setup Data, Basic COCOMO, and Intermediate COCOMO. The second technique is FPA which include several steps, including, Setup Data, Function Point Count and Adjustment Factor. Finally, there is the discussion for the Comparative Studies and Results.

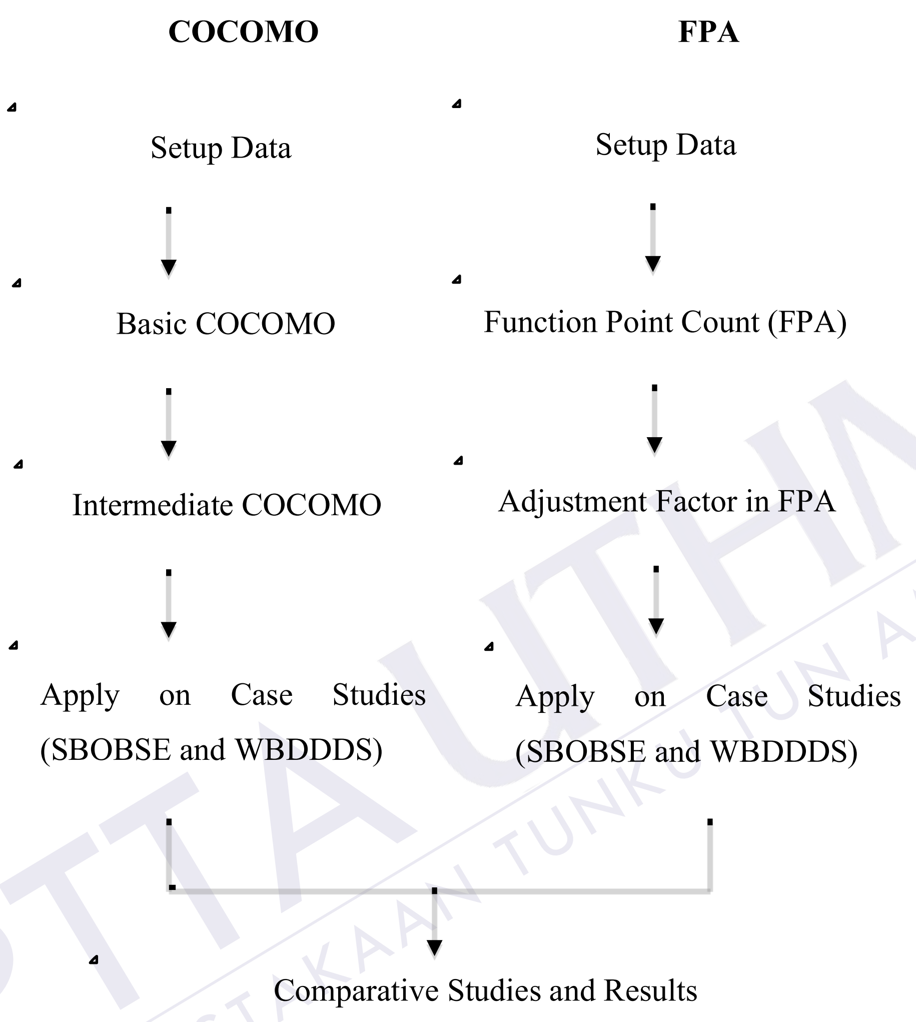
COCOMO
FPA
Setup Data
Setup Data
Basic COCOMO
Function Point Count (FPA)
Intermediate COCOMO
Adjustment Factor in FPA
Apply on Case Studies (SBOBSE and WBDDDS)
Apply on Case Studies (SBOBSE and WBDDDS)
Comparative Studies and Results

Figure 3.1: Flow Chart for Research Activities

Based on Figure 3.1, five phases are needed for each COCOMO and FPA. The software cost was used to apply the estimation of the Constructive Cost Model (COCOMO) technique and the basic COCOMO and Intermediate COCOMO were used in the case study 1 (Sugar Bun Online Bakery System, E-Sugarbun (SBOBSE)). Subsequently, by evaluating the software cost to apply the estimation of the Constructive Cost Model (COCOMO) technique in the case study 2 (Web-Based Dog's Diseases Diagnosis System (WBDDDS)). There are also five steps for FPA to apply this technique. Besides that, the Function Point Analysis (FPA) estimation technique and the Development Project Function Point Count and Adjustment Factor in FPA were used as well in the case study 1 (SBOBSE). The software cost was evaluated to apply the estimation using the Function Point Analysis (FPA) estimation technique in the case study 2 (WBDDDS). Then, the software cost estimation was compared using FPA and COCOMO for both case studies.

3.3 Constructive Cost Model (COCOMO)

The Constructive Cost Model (COCOMO) is the most complete and thoroughly documented model used in cost estimation. This model uses a basic regression formula with parameters that are derived from a historical project data and current project characteristics. COCOMO consists of a hierarchy of two 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 a difference in project attributes of the cost drivers. The Intermediate COCOMO takes these cost drivers into account and additionally accounts for the influence of the individual project phases, as shown in Figure 3.2.

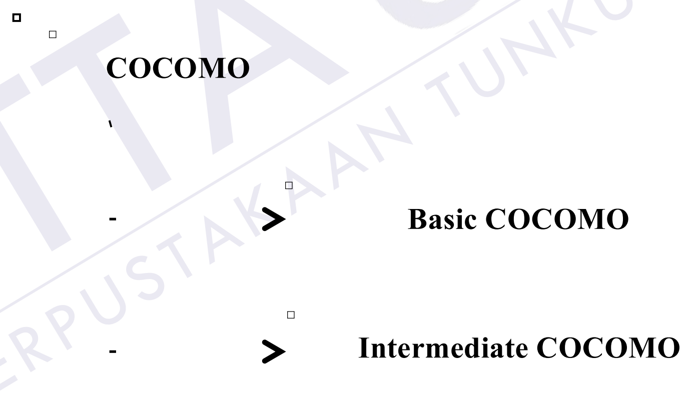

Figure 3.2: COCOMO Increasingly Detailed

3.3.1 Data Setup

Through the data setup for COCOMO, before using the basic COCOMO, the sum of all tasks must be known, and then the KLOC in the project was determined by calculating the number of LOC by the rule. After that, the rule was applied to find all of the required factors (effort applied, development time and people required). All factors were identified based on the analysis of the project. Then, the rating for

COCOMO was finalized to find the total rating of the application of the rules (effort applied, development time and people required).

3.3.2 Basic COCOMO

The COCOMO has three types of projects, which are either organic projects, semi-detached projects or embedded projects. Before starting any project under the term COCOMO, the type of project must be specified. Each project has its own transactions that amount for each category of the software projects: (Organic a_b2.4, b_b1.05, c_b2.5, d_b0.38, Semi-detached a_b3.0, b_b1.12, c_b2.5, d_b0.35 and embedded a_b3.6, b_b1.20, c_b2.5, d_b0.32), where the research is a Semi-detached project. The equation was then applied to calculate the basic COCOMO formula, which consists of the Effort Applied (E), Development Time (D), and People required (P). The Effort Applied (E) is the effort required for people in a month, which is $a_b(KLOC)^{b}{}_{b}$ [person-months] and the calculation of Development Time (D) is c_b(Effort Applied)$^{d}{}_{b}$ [months], and then, the expense of the People required (P) for Execution, divide by Effort Applied/Development Time [count]. But, the KLOC must be calculated before applying the COCOMO formula.

3.3.3 Intermediate COCOMO

The Intermediate COCOMO part of this step, this is an extension of the basic COCOMO model. This estimation model makes use of the set of the cost driver attributes to compute the cost of the software. The intermediate COCOMO computes the software development effort as the function of the program size and a set of cost drivers that include the subjective assessment of the product, hardware, personnel, and project attributes. This extension comprises of a set of four cost drivers, each with a number of subsidiary attributes. Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high". An effort multiplier from Table 3.1 applies to the rating. The product of an all-effort multipliers results in an effort adjustment factor (EAF). The rating will then be used to calculate factors of cost drivers.

Table 3.1 Intermediate COCOMO Coefficients [25]

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

3.3.4 Apply on Case Studies

In this section, the Basic COCOMO and Intermediate COCOMO will be applied on the two case studies of SBOBSE and WBDDDS, where the Basic COCOMO method was applied on two case studies. While COCOMO has three types of projects, which are either organic projects, semi-detached projects or embedded projects. Before starting any project under the term COCOMO, the type of project must be specified. Each project has its own transactions that amount to each category of the software projects: (Organic, Semi-detached and embedded), where then applied to calculate the basic COCOMO formula. The intermediate COCOMO computes the software development effort as the function of the program size and a set of cost drivers that include the subjective assessment of the product, hardware, personnel, and project attributes. This extension comprises of a set of four cost drivers, each with a number of subsidiary attributes. Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high". An effort multiplier from Table 3.1 applies to the rating. The product of all effort multipliers results in an effort adjustment factor (EAF).

3.4 Function Point Analysis (FPA)

The Function Point Analysis (FPA) is an International Organization for Standardization (ISO), which is a recognized method to measure the functional size of an information system. The functional size reflects the amount of functionality that is relevant to be recognised by the user in the business. It is independent of the technology used to implement the system. The unit of measurement is "function points" (fp's). So, FPA expressed the functional size of an information system in a number of function points, for example, the size of a system is 314 fp's. The functional size may be used for the budget application development or enhancement costs and the budget for the annual maintenance costs of the application portfolio as well as to determine the project productivity after completion of the project and to determine the software size for cost estimation.

3.4.1 Data Setup

During the data setup for the FPA Function Point Count, the numbers of externals (inputs, outputs, inquiries, and interfaces) were counted. The first external is the inputs that must identify all inputs of the project to find the external input by the project to determine the extent of file type referenced (FTR) and data element type (DET). The extent to each of the file type was referenced (FTR) and the data element type (DET) was calculated using the recognized rules in the external input. Ultimately, the estimation stage in the External Input was also calculated. The second external is the outputs that must identify all outputs of the project to find the external outputs by the project to determine the extent of file type referenced (FTR) and the extent of data element type (DET). The extensions were calculated using the recognized rules in the external outputs. Finally, the estimation stage in the external outputs was calculated. The third external is the inquiries that must identify all inquiries of the project to find the external inquiry by the project to determine the extent of file type referenced (FTR) and the extent of data element type (DET) using the recognized rules in the external inquiry. The estimation stage in the external inquiry was calculated. The fourth external is the Internal Logical File that must identify all the (ILF) of the project to find the Internal Logical File and by using the project to determine the extent of record element type (RET) and the extent of data element type (DET). The calculation of both extensions of record the element type (RET) and data element type (DET) was done using the recognized rules in the Internal Logical File. Finally, the estimation stage in the Internal Logical File was calculated. As a final point, the external is the interfaces. Subsequently, the FP model was developed to create a list of fourteen general system characteristics that are rated on a scale from 0 to 5 in terms of their likely effect for the system being counted (0 = Not Present, or No Influence, 1 = Incidental Influence, 2 = Moderate Influence, 3 = Average Influence, 4 = Significant Influence and 5 = Strong Influence Throughout). The final AFP number of the system used was compared to the AFP count and the cost of the systems has been measured. The more historical data that can be compared, the better the chances are of accurately estimating the cost of the proposed software system.

3.4.2 Function Point Count

The function points can be counted at all points of a development project from the requirements, including the implementation. This type of count is associated with a new development work. The scope creep can be tracked and monitored by understanding the functional size at all phases of a project. Frequently, this type of count is called a baseline function point count. The function points allow the independence of the underlying language, in which the software is developed.

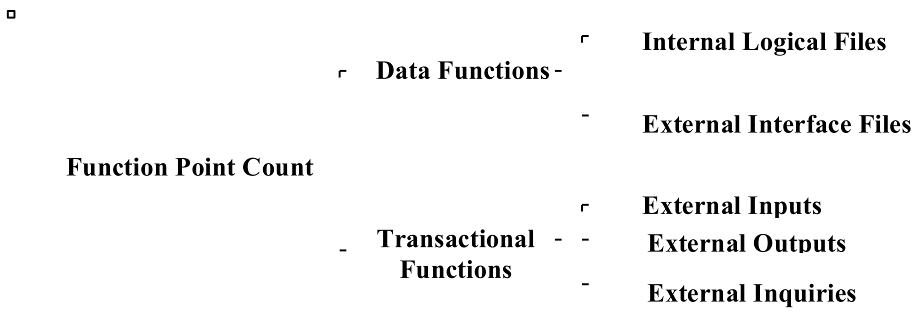

Figure 3.3: Function Point Count

The function points allow the measurement of the software size in standard units and independence of the underlying language, from which the software is developed. Instead of counting the lines of code that make up a system, the number of externals (inputs, outputs, inquiries, and interfaces) is counted, as shown in Figure 3.3. There are five types of externals that were counted. The first type is the external inputs, which are the data or control inputs (input files, tables, forms, screens, messages, etc.) to the system. The second type is the external outputs, which are the data or control outputs from the system. The third type is the external inquiries that are the I/O queries, which require a response (prompts, interrupts, calls, etc.). The fourth type is the external interfaces, which are libraries or programs that are passed into and out of the system (I/O routines, sorting procedures, math libraries, run-time libraries, etc.). Lastly, there are the internal data files, which are groupings of the data stored internally in the system (entities, internal control files, directories). These steps are applied to calculate the size of a project. There is also a count or estimation for all the occurrences of each type of externals. Each occurrence is assigned a complexity weight and after that, each occurrence is multiplied by its complexity weight. In total, the results will obtain a function count. The complexity weights are listed in Table 3.2, and the function count is multiplied by the value adjustment multiplier (VAM) to obtain the function point count.

Table 3.2 Complexity Weights [32]

	Complexity		
Description	Low	Average	High
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External interfaces	5	7	10
Internal files	7	10	15

3.4.3 Adjustment Factor in FPA

Although the Adjustment Factor (AF) can give us a good idea of the number of functions in a system, it doesn’t take into account the environment variables for determining the effort required to program the system. For example, a software system that requires a very high performance would require an additional effort to ensure that the software is written as efficiently as possible. Albrecht [25] recognized this when developing the FP model and he created a list of fourteen "general system characteristics that are rated on a scale from 0 to 5 in terms of their likely effect for the system being counted." These characteristics are as the following in Table 3.3.

Table 3.3 Value Adjustment Factor [32]

GSCS's Factors	Rank					
	0	1	2	3	4	5
Data Communications	No Influence	Incidental	Moderate	Average	Significant	Essential
Distributed Functions						
Performance						
Heavily Used Configuration						
Transaction Rate						
Online Data Entry						
End User Efficiency						
Online Update						
Complex Processing						
Reusability						
Installation Ease						
Operational Ease						
Multiple Sites						
Facilitate Change						

In practice, the final AFP number of the proposed system is compared against the AFP count and the cost of systems that have been measured in the past. The more historical data that can be compared, the better the chances are at accurately estimating the cost of the proposed software system. To continuously refine the estimation accuracy, it is essential that the actual cost is measured and recorded once a system has been completed. It is this actual cost that enables the evaluation of the initial estimate.

3.4.4 Apply on Case Studies

During the application in this section, the FPA will be applied on the two case studies of SBOBSE and WBDDDS. For The FPA Function Point Count, the numbers of externals (inputs, outputs, inquiries, and interfaces) are counted. There are also steps that are applied to calculate the size of a project. There is also a count or estimation for all the occurrences of each type of externals. Each occurrence is assigned a complexity weight and after that, each occurrence is multiplied by its complexity weight.

3.5 Comparative Studies and Results Discussion

This section explains how the comparison is performed using COCOMO on both case studies (Sugar Bun Online Bakery System, E-Sugarbun (SBOBSE) and Web-Based Dog's Diseases Diagnosis System (WBDDDS)) and using the FPA on both case studies. COCOMO has two techniques (Basic COCOMO and Intermediate COCOMO) that were applied to the case studies. As mentioned in Figure 3.1, the FPA has two parameters that are the Function Point Count and the Adjustment Factor in FPA where the function point count has five parameters that are External Input (EI), External Outputs (EO), External Inquiry (EI), Internal Logical File (ILF) and External Interface File that were applied to the case studies. In addition, after completing the data collection and analysis, the man-months and total cost in each project must be estimated. After the majority of the requirements are found, the comparison between each of the man-months and total cost was performed to find out which costs is lesser and which one had lesser man-months.

3.6 Summary

This chapter has discussed the methodology used for the FPA with five parameters followed by the calculation of unadjusted function point counts and adjustment factor in the FPA. The COCOMO was used with two parameters calculated to get the formula. The next chapter will look further on the two techniques of the FPA and COCOMO, based on the methodology proposed in this chapter.

REFERENCES

1. Attarzadeh, I., & Ow, S. H. (2010). A novel soft computing model to increase the accuracy of software development cost estimation. *The 2nd International Conference.* In Computer and Automation Engineering (ICCAE), 2010 pp. 603 - 607.
2. Boehm, B. W. (1979). Software engineering-as it is. *In Proceedings of the 4th International Conference on Software Engineering* (pp. 11-21). IEEE Press.
3. Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., & Selby, R. (1995). *Cost models for future software life cycle processes*: COCOMO 2.0.Annals of software engineering, 1(1), pp. 57 - 94.
4. Standish (1994) *Project Success Safee Improves Every 10 years*. Technical Report. Standish Group.
5. Matson, J. E., Barrett, B. E., & Mellichamp, J. M. (1994). Software development cost estimation using function points. Software Engineering, *IEEE Transactions on*, 20(4), pp. 275 - 287.
6. Felfernig, A. and Salbrechter, A. (2004). Applying function point analysis to effort estimation in configurator development. *In International Conference on Economic, Technical and organisational aspects of Product Configuration Systems, Kopenhagen, Denmark,* pp. 109 - 119.
7. Junhai M. and Lingling M. (2010), Comparison Study on Methods of Software Cost Estimation. The 2nd International Conference on e-Business and Information System Security (EBISS), pp. 1-4.
8. Jeng, B., Yeh, D., Wang, D., Chu, S. L., & Chen, C. M. (2011). A Specific Effort Estimation Method Using Function Point. *Journal of Information Science and Engineering,* 27(4), pp. 1363 - 1376.
9. Mittas, N. and Angelis, L. (2013). Overestimation and Underestimation of Software Cost Models: Evaluation by Visualization. In Software Engineering

and Advanced Applications (SEAA), *2013 39th EUROMICRO Conference on* (pp. 317-324). IEEE.

10. Moløkken-Ostvold, K., Jorgensen, M., Tanilkan, S. S., Gallis, H., Lien, A. C. & Hove, S. W. (2004). A survey on software estimation in the Norwegian industry. In Software Metrics, 2004. Proceedings. *10th International Symposium on* (pp. 208-219). IEEE.
11. Nasir, M. (2006). A survey of software estimation techniques and project planning practices. In Proceeding of the Seventh ACIS Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2006. pp. 305-310.
12. Boehm, B. W., Madachy, R., & Steece, B. (2000). *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall.
13. Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach*, 7/e, Pressman & Associates: Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020.
14. Pandian, C. R. (2004). Software Metrics: A Guide to Planning. Analysis, and Application. Auerbach Publications.
15. McConnell, S. (1998). Software Project Survival Guide. Microsoft Press, 1998. pp. 40-45, Nov./Dec. 2006, doi:10.1109/MITP.2006.149.
16. Tagra, D. (2011). *Cost Estimation For Commercial Software Development Organizations*. Dalhousie University Halifax, Nova Scotia: Ph.D. Thesis.
17. K. Kavoussanakis and T. Sloan. UKHEC Report on Software Estimation, Dec 2004
18. Laird, L. M. (2006). The limitations of estimation. IT professional, 8(6), 40-45.
19. Khatibi, V. & A.Jawawi, Dayang N. (2010). Software Cost Estimation Methods: A Review. *Journal of Emerging Trends in Computing and Information Sciences,* 3(1), pp. 21 – 29.
20. Boehm, B. (2000). Safe and simple software cost analysis. IEEE software, 17(5), pp. 14-17
21. Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. *Annals of Software Engineering,* 10(1-4), pp. 177-205.

22. Kumari, S. and Pushank, S. (2013). Performance analysis of the software cost Estimation Methods: A Review. *International Journal of Advanced Research in Computer Science and Software Engineering,* 3(7)*,* (pp. 229 – 238).
23. H. Leung and Z. Fan, (2002). Software Cost Estimation. Handbook of software engineering and knowledge engineering, *World Scientific Publications Company*. pp. 1-14.
24. Abts, C., Boehm, B. W., & Clark, E. B. (2000). COCOTS: A COTS software integration lifecycle cost model-model overview and preliminary data collection findings. *In ESCOM-SCOPE Conference*.
25. Albrecht, A. J. (1979). Measuring application development productivity. In Proceedings of the *Joint SHARE/GUIDE/IBM Application Development Symposium*. pp. 83-92.
26. Longstreet, D. (2002). Fundamentals of function point analysis. Blue Springs: Longstreet Consulting.
27. Ajmera, R., Sinha, R. R., & Lamba, C. S. (2012). Comparative analysis of Software testing measurement technique. *International Journal of Engineering and Innovative Technology (IJEIT),* 1(2), pp. 70 – 80.
28. Patel, S. (2013). Function point distribution using maximum entropy principle. In Image Information Processing (ICIIP), *2013 IEEE Second International Conference on* (pp. 684-689). IEEE.
29. Sadiq, M., Zafar, S., Asim, M., & Suman, R. (2010). GUI of esrcTool: A Tool to Estimate the Software Risk and Cost, The 2nd IEEE International Conference on Computer and Automation Engineering (ICCAE-2010), Singapore, pp 673-677.
30. Wu, J., and Cai, X. (2008). A software size measurement model for large- scale business applications, in Proceedings of the 2008 International Conference on Computer Science and Software Engineering (CSSE), 2008. (Washington, DC, USA), pp. 39–42.
31. Kemerer, C. F. (1987). An empirical validation of software cost estimation models. Communications of the ACM, 30(5), pp. 416-429.
32. Sheta, A., and Aljahdali, S. (2013). Software Effort Estimation Inspired by COCOMO and FP Models: A Fuzzy Logic Approach. *International Journal of Advanced Computer Science and Applications*. 4(11), pp. 192 – 197.

33. Heemstra, F.J. and Kusters, R.J. (1991). Function point analysis: Evaluation of a software cost estimation model. *European Journal of Information Systems.* 1(4). pp. 223-237.
34. Lederer, A.L. and Prasad, J. (1993). Information systems software cost estimating: a current assessment. *Journal of Information Technology,* 8(1). pp. 22-33.