ISSN 2277-3061

Performance Evaluation of COCOMO Vs UCP

Er. Monika Dureja
Department of Computer Science, CDLU Sirsa
er.monika.dureja@gmail.com

ABSTRACT

Effort estimation is a measure factor of estimation in today's world. From a long time being, a no. of techniques is employed for the estimation of efforts. Very popular and widely used techniques are COCOMO1 and COCOMOII that estimates the cost in terms of efforts. But effort is linearly or non-linearly dependent upon the size of the developing software. For size estimation, Lines of Code (LOC) counts technique showed its comforts as developer do not have to solve any mathematical equations, just count the no. of lines of the developed software. But uncertainties in its results and other limitations led us to use different approaches. Then came Functions Point (FP) technique, this technique proved its goodness in almost every aspect where LOC lacks. Developers estimate Lines of code early before the software is finally built. Several factors have to be considered before estimation as how many no. of input screens, output screens, external inquiries, and database related internal logical files, external interface files and many other technical factors that affect accuracy of estimation. FP had been used for a long time being for estimation of size, which is the main input for COCOMO1 and COCOMOII.

But as the time changes, needs and their solutions also changes. Today, software are based on object oriented paradigm and OOP languages. Developers use Unified Modeling Language (UML) notations and diagrams for estimation of each aspect of software development. So the main factor is to use such a technique that supports the OOPs for estimation purposes. Main approach of this thesis is to implement Use Case Point estimation (UCP) Technique to overcome the drawbacks of FP which was considered as a procedural oriented (POP).

Further evaluation of the performance of UCP in comparative to COCOMO 1 and COCOMOII is taken. This all will be performed with the help of a self implemented tool having all the functionalities at one location.

Keywords: cococmo; sdlc; fp; ucp; kloc.

INTRODUCTION

Estimates of cost and schedule in software projects are based on a prediction of the size of the future system. Unfortunately, the estimation of cost and schedule rarely comes within budget and on-time. Many elements of insecurity affect the estimation of efforts. Reliable early estimates are difficult to obtain because of the lack of detailed information about the future system at an early stage. However, early estimates are required only when there is any kind of contract making about the development of software for any kind of problem. Secondly, when we are to determine whether a project is feasible to make or not in terms of Cost-Benefit analysis [1].

Traditional cost models may take software size as an input parameter, and then apply a set of adjustment factors or 'cost drivers' to compute an estimate of total effort. In Object oriented software production, use case describes functional requirements. The use case model is used to predict the size of the future software system at an early development stage.

This dissertation work describes the various estimation methods combined into a single tool. Tools consists estimation models of main input parameter size. These are Source Lines Of Code (SLOC) and Function Point (FP).

This Size Input is incorporated into cost estimation models like COCOMO I and COCOMO II from where we estimate other factors like Effort, duration, Cost etc. But the main aim of this tool is to provide better estimates of size for this time trend languages which are based on Object Oriented Paradigm. To better describe the functional requirements of OOP paradigm that are based on use case modeling, Use case point estimation technique is used.

We will now briefly explain the steps in the use case point's method as used by Karner (Karner 93.) First, categorize the actors in the use case model as simple, average or complex and calculate the total *unadjusted actor weight (UAW)* by counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the points. Next, categorize the use cases as simple, average or complex, depending on the number of transactions, including the transactions in alternative flows. Then the *unadjusted use case weights (UUCW)* are calculated by counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The *UAW* is added to the *UUCW* to get the *unadjusted use case points (UUCP)* [2].

Next, the use case points are adjusted based on the values assigned to a number of technical factors and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the overall project

METHODOLOGY

COCOMO1 (COnstructive COst MOdel):

Original COCOMO stands for Constructive cost model. The word "constructive" implies the openness of the model because while using COCOMO model for estimation we can easily show the reasons of estimated results [4].

The model was first introduced by Dr. Barry Boehm in 1981, Developed at TRW, and a US defense contractor.

The COnstructive COst MOdel (COCOMO) is an algorithmic Software cost model that uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics. A survey on 63 projects ranging from 2,000 to 10,000 lines of code, and programming languages ranging from assembly to PL/I were conducted and on the basis of their mutual outputs, the model was framed and the name given to that model was COCOMO I. Due to changing environments of software and hardware developments, many revisions have been performed on original COCOMO [11].

The 63 projects were based on the waterfall model which was the basis for software development process in 1981. So COCOMO estimates rely on waterfall model.

COCOMO I is a simple on-line available cost estimation model which is used for the estimation of efforts or person-months required to develop software. Other factors like duration, productivity, cost estimations are also can be derived through COCOMO I.

COCOMO is a hierarchy of software cost estimation models which include three basic, intermediate and detailed sub models and three modes organic, semi-detached and embedded [5].

COCOMO I depend on two main equations [4]:

Development Effort: $MM= a*KDSI^b$ (1)

Based on MM – Man-Months/Person-months/Staff months is one month of effort by one person.

Default value for Effort is 152 hours per Person months.

Duration: $TDEV=2.5*MM^c$ (2)

The coefficients a, b and c are constants and their values depend on the mode of the development

Modes of development are [5]:

1. Organic Mode: Typical 2-50 KLOC sized projects come in this mode category. Usually a small team of experienced developers develops the software in a Familiar and in-house environment where no tight deadline exists. For e.g., payroll, Inventory projects etc.

2. Semi- Detached Mode: Typically 50-300 KLOC sized projects come in this mode category. A medium size team of average experience developers develops the software in medium constraints of environment and deadline. For e.g., Utility systems like compilers, Database systems, editors etc.
3. Embedded Mode: Typically over 300KLOC sized projects come in this mode category. A team of very little previous experience developers develop the software's of Real time systems and complex interfaces where a tight deadline exists. For e.g., ATMs, Air Traffic control etc.

A static single-valued model that computes software development effort and cost as a function of program size expressed in estimated thousand delivered source instructions (KDSI).

The Basic COCOMO applies the parameterized equation without consideration of project characteristics.

$$E= a*KDSI^{b}$$
$$D=2.5* MM^{c}$$

Where E is the Effort per Person-Months and D is the Development time in months. The coefficients a, b and c are given in table.

Basic COCOMO	a	B	C
Organic	2.4	1.05	0.38
Semi-Detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Table 1: Parameters of COCOMO1 Basic Model [4]

When Effort and Development Time are known, the average staff size to complete the project may be calculated as [5]:

Average Staff Size = E/D persons.

When project size is known, the productivity level may be calculated as:

Productivity (P) = KDSI/E

Productivity measure is in KLOC/PM

COCOMO II:

Need is the mother of invention. This phrase proofs its truthiness in this case also. Problems with COCOMO1 lead to the formation of COCOMOII model. COCOMO1 was introduced in 1981. But in late 90's, COCOMO 1 faced a lot of problems in estimation of new life cycle processes such as non-sequential and rapid development process models, reuse-driven approaches, and object-oriented approaches [4]. COCOMO II considers all these factors with some improvements in COCOMO1. COCOMO II was published initially in the Annals of Software Engineering in 1995 with three sub models; an application-composition model, an early design model and a post-architecture model.

FIVE SCALE FACTORS:

A detailed study conclude that most significant input to the COCOMO II model is size, so, a good size estimate is very important for any good model estimation. Size in COCOMO is treated as a special cost driver, so it has an exponential factor, E. The exponent E in effort is an aggregation of five scale factors. All scale factors have rating levels. These rating levels are Very Low (VL), Low (L), Nominal (N), High (H), Very High (VH) and Extra High (XH). Each rating level has a weight, W, which is a quantitative value used in the COCOMO model. The five COCOMO II scale factors with their description and ratings are shown in table:

Description	Scale factor	Very low (0.05)	Low(0.04)	Nominal(0.03)	High(0.02)	Very high(0.01)	extra high (0.00)
Precedentedness	PREC	Thoroughly Unprecedented	Largely Unprecedented	Somewhat Unprecedented	Generally Familiar	Largely Familiar	Thoroughly Familiar
Flexibility	FLEX	Rigorous	Occasional relaxation	Some relaxation	General Conformity	Some conformity	General goals
Risk Resolution	RESL	Little (20%)	Some (40%)	Often (60%)	Generally (75%)	Mostly (90%)	Full (100%)
Team cohesion	TEAM	Very difficult interaction	Some difficult	Basically cooperative interactions	Largely cooperative	Highly cooperative	Seamless Interaction
Process Maturity	PMAT	Weighted average of "yes" Answers to CMM maturity Questionnaire					

Table 2: COCOMO II Scale Factors

To calculate the time duration:

Duration TDEV: $\boxed{TDEV_{NS}=C \times (PM)^{F}}$ where $F= D+0.2x0.01x\sum SF_j$

Or Where $F = D + 0.2 \times (E-B)$

And C = 3.67 and D = 0.28

When Effort and Development Time are known, the average staff size to complete the project may be calculated as:

Average Staff Size = E/D persons.

When project size is known, the productivity level may be calculated as:

Productivity (P) = KDSI/E

Productivity measure is in KLOC/PM.

The Early Design and Post-Architecture model use the same approach for product sizing (including reuse) and for scale factors as well as for estimating the amount of effort (PM) and calendar time it will take to develop (TDEV) a software project. It uses source lines of code or function points for sizing, a set of 17 Effort Multipliers(EMs) and a set of 5 scaling cost drivers to determine the project's Scaling Factors (SFs). The formula for post architecture model is same as of early design model with just difference in total effort multipliers used and their rating

Cost Factors	**Description**	**Rating**				
		Very low	**Low**	**Nominal**	**High**	**Very high**
RELY	Required Software Reliability	0.82	0.92	1.00	1.10	1.26
DATA	Data base Size	0.00	0.90	1.00	1.14	1.28
RUSE	Developed for Reusability	0.00	0.95	1.00	1.07	1.15
DOCU	Documentation needs	0.81	0.91	1.00	1.11	1.23
CPLX	Product complexity	0.73	0.81	1.00	1.17	1.34
TIME	Execution Time Constraints	0.00	-	1.00	1.11	1.29
STOR	Main Storage Constraints	0.00	-	1.00	1.05	1.17

PVOL	Platform Volatility	0.00	0.87	1.00	1.15	1.30
ACAP	Analyst Capability	1.42	1.19	1.00	0.85	0.71
PCAP	Programmer Capability	1.34	1.15	1.00	0.88	0.76
APEX	Application Experience	1.22	1.10	1.00	0.88	0.81
PLEX	Platform Experience	1.19	1.09	1.00	0.91	0.85
LTEX	Language ad Tool Experience	1.20	1.09	1.00	0.91	0.84
PCON	Personnel Continuity	1.29	1.12	1.00	0.90	0.81
TOOL	Use of Software Tools	1.17	1.09	1.00	0.90	0.78
SITE	Multi site Development	1.22	1.09	1.00	0.93	0.86
SCED	Required Development Schedule	1.43	1.14	1.00	1.00	1.00

Table 3: COCOMO II Post-Architecture Model Cost Drivers [10]

Scale factors description is already shown in table

Duration TDEV: $C^* (PM)^F$ where $F= D+0.2x0.01x\sum SF_j$

Or Where $F = D + 0.2 x (E-B)$

And C = 3.67 and D = 0.28

When Effort and Development Time are known, the average staff size to complete the project may be calculated as:

Average Staff Size = E/D persons.

When project size is known, the productivity level may be calculated as:

Productivity (P) = KDSI/E

Productivity measure is in KLOC/PM.

USE CASE POINT ESTIMATION TECHNIQUE

This technique was proposed by Gustav Karner in 1993 to estimate the projects based on OO. Some little changes were being made in the technique, but the main concept is used as it is in afterward extensions.

Use case point estimation is calculated from use case model. The main aspects of use case point estimate technique are actors, use cases, associations between actors and use cases, relationships between actors, relationships between use cases. UCP is measured by counting the number of actors and transactions included in the flow of events with some weight. A transaction is an event that occurs between an actor and the target system, the event being performed entirely or not at all [22].

The UCP counting process consists of the following steps [25]:

1. Unadjusted Actors Weights (UAW)
2. Unadjusted Use Case Weights (UUCW)
3. Unadjusted Use Case Points (UUCP)
4. Technical Complexity Factor (TCF)
5. Environmental Factor (EF)
6. Adjusted Use Case Points (AUCP)
7. Staff hours per Use Case Point.

Step 1 - Unadjusted Actors Weights (UAW)

Categorization of actors in the model is as it can be a simple, average or a complex actor depending on the way of its interaction with the system. Each actor possesses some weight and lastly their no. multiplied sum is termed as Unadjusted Actors Weight.

A Simple actor can be represented by another system with some defined interfaces or it can be another system interacting through API (Application Programming Interface). It possesses weight **1**. This can be calculated by using the following formula:

SActor = Σ (SimpleActor) * SimpleWeight

Where SActor stands for Simple Actor

The Average actor can be represented by a system interacting through protocols such as TCP/IP or it is a person interacting through a text-based interface (such as an old ASCII terminal). It possesses weight **2**, and can be obtained from the following formula:

AActor = Σ (AverageActor) * AverageWeight

Where AActor stands for Average Actor

The Complex actor can be represented by a person that interacts through graphical interfaces or Internet. It possesses weight **3**. The following formula gives that number.

CActor = Σ (ComplexActor) * ComplexWeight

After one get done all calculations previous presented, the UAW can be obtained by:

UAW = SActor + AActor + CActor

Weight Factor	Type	Description
1	Simple	Program Interface
2	Average	Interactive, protocol driven Interface
3	Complex	Graphical User Interface

Table4: Classification of actors

Step 2 - Unadjusted Use Case Weights (UUCW)

The same process used for actors, should be repeated with the use cases. That process should rate use cases among Simple, Average and Complex.

If the use case possesses up to 3 transactions, it should be considered as a simple use case (SUsecase), therefore weight is equal to **5**. That can be calculated by:

SUsecase = Σ (SimpleUC) * SimpleWeight

On the other hand, if the use case possesses from 4 to 7 transactions, it should be considered as an average use case (AUsecase), therefore weight is equal to **10**. That can be calculated by:

AUsecase = Σ (AverageUC) * AverageWeight

Concluding, if the use case possesses more than 7 transactions, it should be considered as a complex one (CUsecase), therefore weight equals **15**. That can be calculated by:

CUsecase = Σ (ComplexUC) * ComplexWeight

After get done all calculations previously presented, the UUCW can be obtained from:

UUCW = SUsecase + AUsecase + CUsecase

Type	Description	Weight factor
Simple	Less than 3 transactions	5
Average	4 to 7 transactions	10
Complex	More than 7 transactions	15

Table 5: Transaction-Based Weighting Factor

Step 3 - Unadjusted Use Case Points - UUCP

Unadjusted Use Case Points (UUCP) is simply the sum of UAW and UUCW. This can be expressed by:

UUCP = UAW + UUCW

Step 4- Technical Complexity Factor (TCF)

Next, the use case points are adjusted based on the values assigned to a number of technical factors and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the overall project [35].

Factor no.	Description	Weight
T1	Distributed system	2
T2	Response or Throughput performance objective	1
T3	End-user efficiency	1
T4	Complex Internal processing	1
T5	Code must be reusable	1
T6	Easy to Install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes Special Security Features	1
T12	Provides Direct Access for Third Parties	1
T13	Special user Training facilities are Required	1

Table 6: Technical Complexity Factor weighting for UCP

The Technical Complexity Factor (TCF):

TCF = 0.6 + (.01 * Tfactor)

Where Tfactor is the sum of multiplication of weight with the no. of factors

Step 5: Environmental Factor:

The level of the environmental complexity is directly related to professionals' experience involved in the project. The environmental factors vary in a scale from 0 to 5, indicating little and large experience.

EF is calculated considering Table

Factor	**Description**	**Weight**
F1	Familiar with Process	1.5
F2	Application experience	0.5
F3	Object oriented experience	1.0
F4	Analyst capability	0.5
F5	Motivation	1.0
F6	Stable requirements	2.0
F7	Part time workers	-1.0
F8	Difficult programming language	-1.0

Table 7: Environmental Complexity Factors

The Environmental Factor (EF):

EF = 1.4 + (-0.03 * Efactor)

Where Efactor is the sum of multiplication of weight with the no. of factors

Step 6 - Adjusted Use Case Points (AUCP)

The Adjusted Use Case Points (AUCP) is calculated considering UUCP, TCF, and EF from the following formula:

AUCP = UUCP * TCF * EF

Finally, the UCP is multiplied by a historically collected figure representing productivity, such as a factor of 20-staff hrs per use case point, to arrive at a project estimate. The result is an estimate of the total number of person hours required to complete the project.

Step 7 - Staff Hours per Use Case Point

The spent time for programming per person needed to accomplish each one of AUCP should be calculated. Karner proposed the use of 20person-hour for each AUCP unit.

Duration = AUCP* PF person hours

Where Productivity Factor (PF) = 20.

Effort =Duration/152 person-months

Cost = (labor rate)* effort.

Conclusion

The tool will help in requirement gathering in object oriented paradigm. The result shows that effort and duration is less comparative to the COCOMO 1and COCOMO II. The cost will reduce accordingly as efforts

Object-oriented design and development are popular concepts in today's software development environment.

In conclusion, use case points method of effort estimation is a very valuable addition to the tools available for the project manager. The method can be very reliable or just as reliable as other effort estimation tools such as COCOMO, function point and lines of code. All of the estimation methods are susceptible to error, and require accurate historical figures for productivity in order to be useful within the context of the organization. Use case points method is especially valuable in those system development projects where use cases are produced anyway. It is comparable to the function point method that has become quite respected throughout the industry. It provides estimates that are sometimes better than what experts can provide to the industry. Expert estimates should not be excluded from the process of estimation; rather it should be used in conjunction with use case estimates to ensure an accurate estimate. Lastly, with standardization and kind of national and international efforts that have helped the function point method become widely accepted, this method also has the potential to become a mature and widely accepted estimation tool.

Future Scope

There have been a lot of works that can be done as a future scope. There can be improvement in implemented tool. There is no way to calculate the cost in UCP technique, although the method given by Smith is helpful up to some extent but it’s much complicated. So there should be a direct way to calculate cost from efforts from UCP technique. A conversion from Use Case Points to Function Points should be there, as historical data play a vital role for estimation of future system. FP has been used from a long time and there exists a lot of historical records related to FP. And UCP is still a new approach. So if a conversion from UCP to FP is find out, it will be a great research for this time.

References

1) Gautam Banerjee, *“Use Case Points- An Estimation Approach”* Aug. 2001.

2) Mel Damodaran, ***“Estimations using Use Case Points”***

3) Nancy Merlo – Schett, ***“Seminar on Software Cost Estimation”**,* Computer Science, 2002

4) Rajiv aggarwal book. K k aggarwal.

5) Costar tool table.

6) Shinji kusumoto, Fumikazu matukawa, Katsuro inoue, **“Effort Estimation Tool Based on Use Case Points Method”,**

7) Caio Monteiro Barbosa da Silva, Denis Silva Loubach and Adilson Marques da Cunha, “**APPLYING THE USE CASE POINTS EFFORT ESTIMATION TECHNIQUE TO AVIONICS SYSTEMS”** 27th Digital Avionics Systems Conference pp 5.B.4-1 to 5.B 4-10, 2008.

8) Karner, G. Metrics for Objectory. Diploma thesis, University of Link6ping, Sweden. No. LiTH-IDA-Ex-9344:21, December 1993.

9) B. Anda, H. Dreiem, D.I.K. Sjøberg, M. Jorgensen, **"Estimating Software Development Effort based on Use Cases - Experiences from Industry",** Proc. of Fourth International Conference on the UML, pp. 487-504(2001).

10) José Antonio Paw-Sang, Enrique Jolay-Vasquez ***"An Approach of a Technique for Effort Estimation of Iterations in Software Projects"*** IEEE, 2006, pg no.

11) Rumbaugh, I., Jacobson, I., Booch, G.*"****Unified Modeling Language Reference Manual"*** Addison Wesley, 1997.

Monika Dureja PH.d Shcolar at CDLUSirsa, Computer science Department Received M.Tech Degree from UIET, KUK. She Received B.Tech Degree from JCDCOE, SIrsa.