Software Cost Estimation Using Constructive Cost Model (COCOMO II)

Kyaw Thu Thein, Tar Tar Khin
Computer University, Pyay, Myanmar
kyawthuthein@gmail.com, cupyay2009@gmail.com

Abstract

Software cost estimation is an important role in software engineering practice, often determining the success or failure of contract negotiation and project execution. Cost estimation's deliverables such as efforts, schedules, and staff requirements are valuable informations for project formation and execution. Thus, the software engineering community has put tremendous efforts to develop models that can help estimators to generate the accurate cost estimate of a software project. Software cost estimation systems are used for software development efforts, software costs and software schedule estimations.

Keywords: Software Effort Estimation, Functional Size Measurement, COCOMO II, Cost Drivers

1. Introduction

Over the years, software managers and software engineers have used various cost models such as the Constructive Cost Model (COCOMO) to support their software cost and estimation processes.

COCOMO (Constructive Cost Model) is a model that allows software project managers to estimate project cost and duration. It was developed initially (COCOMO 81) by Barry Boehm. These models have also helped them to reason about the cost and schedule implications of their development decisions, investment decisions, client negotiations and requested changes, risk management decisions, and process improvement decisions.

Since that time, COCOMO has cultivated a user community that has contributed to its development and calibration.

The COCOMO II model is a COCOMO 81 update to address software development practices in the 1990's and 2000's. This eventually led to the current version of the model: COCOMO II.2000.3. COCOMO has also evolved to meet user needs as the scope and complexity of software system development has grown. The growing need for the model to estimate different aspects of software development served as a catalyst for the creation of derivative models and extensions that could better address commercial off-the-shelf software integration, system engineering, and system-of-systems architecting and engineering.

2. Background

Software cost estimation means to predict the effort required to develop a software. In software cost estimation, we predict that how much effort and time is needed to complete the software. Software cost estimation helps us to determine what resources we should use to complete the projects on time. It also helps us in better software development planning.

2.1. COCOMO II

COCOMO II combines the COCOMO and Ada COCOMO scaling approaches into a single rating-driven model. It is similar to that of Ada COCOMO in having additive factors applied to a base exponent *E*. It includes the Ada COCOMO factors, but combines the architecture and risk factors into a single factor, and replaces the Ada process maturity factor with a Software Engineering Institute (SEI) process maturity factor (The exact form of this factor is still being worked out with the SEI). The scaling model also adds two factors, precedentedness and flexibility, to account for the mode effects in original COCOMO, and adds a Team Cohesiveness factor to account for the diseconomy-of-scale effects on software projects whose developers, customers, and users have difficulty in synchronizing their efforts. It does not include the Ada COCOMO Requirements Volatility factor, which is now covered by increasing the effective product size via the Breakage factor.

3. Function Points and Size Estimation

3.1. Source Line of Code

The COCOMO calculations are based on

estimate of project's size measured in Source Lines of Code (SLOC). A Source Line of Code (SLOC) is defined such that:

- Source lines that are delivered as part of the product are included – test drivers and other support software is excluded.
- Source lines are created by the project staff – code generated by applications generators is excluded.
- One line of code is one logical statement.
- Declarations are counted as lines of code.
- Comments are not counted as lines of code.

3.2. Function Point

Function points are based on a combination of program characteristics. The approach is to identify and count a number of unique function types:

External Input (Inputs)	Count each unique user data or user control input type that (i) enters the external boundary of the software system being measured and (ii) adds or changes data in a logical internal file.
External Output (Output)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical Files (Files)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (Interface)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (Queries)	Count each unique input-output combination, where an input causes and generates an immediate output, as an external inquiry type.

3.2.1. Function Point Complexity Weight

Apply complexity weights. Weight the number in each cell using the following scheme. The weights reflect the relative value of the function to the user.

	Simple	Average	Complex
Internal Logical Files	7	10	15
External Interface Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External inquiries	3	4	6

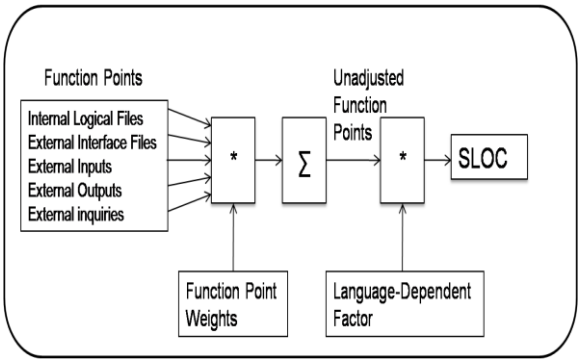
3.3. Language Dependent Factors

To determine the nominal person months for the Early Design model, the unadjusted function points have to be converted to source lines of code in the implementation language (assembly, higher order language, fourth-generation language, etc.) in order to assess the relative conciseness of implementation per function point.

Language	SLOC / UFP	Language	SLOC / UFP
Access	38	Third Generation Language	80
Assembly - Basic	320	Fourth Generation Language	20
Basic - Visual	32	Fifth Generation Language	5
C	128	High Level Language	64
C++	55	HTML 3.0	15
Cobol (ANSI 85)	91	Visual Basic 5.0	29
First Generation Language	320	Visual C++	34
Second Generation Language	107	Java	53

3.4. Size Estimation

A good size estimate is very important for a good model estimation. However, determining size can be challenging. Projects are generally composed of new code, code reused from other sources--with or without modifications--and automatically translated code. COCOMO II only uses size data that influences effort which is new code and code that is copied and modified.



4. Scale Factors and Cost Drivers

4.1. Scale Factors

The selection of scale factors is based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation.

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
Precedented-ness (PREC)	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally Familiar	largely familiar	thoroughly Familiar
Development flexibility (FLEX)	rigorous	occasional Relaxation	some Relaxation	general Conformity	some Conformity	general goals
Architecture/ risk resolution (RESL)	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
Team cohesion (TEAM)	very difficult Interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
Process maturity (PMAT)	SW-CMM Level 1 lower	SW-CMM Level 1 upper "initial"	SW-CMM Level 2 "repeatable"	SW-CMM Level 3 "defined"	SW-CMM Level 4 "managed"	SW-CMM Level 5 "optimising"

Each scale factor has a range of rating levels, from Very Low to Extra High. Each rating level has a weight, *SF*, and the specific value of the weight is called a scale factor. A project's scale factors, *SF*i, are summed across all of the factors, and used to determine a scale exponent, *E*, via the following formula:

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

$$\text{where } B = 0.91 \text{ (for COCOMO II.2000)}$$

4.2. Cost Drivers

The 17 Post-Architecture cost drivers are used in the COCOMO II model to adjust the nominal effort, Person-Months, to reflect the software product under development. Each cost driver is defined below by a set of rating levels and a corresponding set of effort multipliers. The Nominal level always has an effort multiplier of 1.00, which does not change the estimated effort. Off-nominal ratings generally do change the estimated effort.

The cost drivers that are used to adjust the initial estimates in the post-architecture model fall into four classes:

1. ***Product cost drivers*** are concerned with required characteristics of the software product being developed.
2. ***Computer cost drivers*** are constraints imposed on the software by the hardware platform.
3. ***Personnel cost drivers*** are multipliers that take the experience and capabilities of the people working on the project into account.
4. ***Project cost drivers*** are concerned with the particular characteristics of the software development project.

4.2.1 Product cost drivers

RELY	Required software reliability
CPLX	Product complexity
DOCU	Documentation match to life-cycle needs
DATA	Database size
RUSE	Required reusability

4.2.2 Computer cost drivers

TIME	Execution time constraint
STOR	Main storage constraint
PVOL	Platform volatility

4.2.3 Personnel cost drivers

ACAP	Analyst capabilities
PCAP	Programmer capabilities
AEXP	Applications experience
PEXP	Platform experience
LTEX	Programming language experience
PCON	Personnel continuity

4.2.4 Project cost drivers

TOOL	Use of software tools
SITE	Multisite development
SCED	Required schedule development

4.3. COCOMO II Version Parameter Values

	Scales Factor Values					
	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00
	Cost Drivers Values					
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.05	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	
PCON	1.29	1.12	1.00	0.90	0.81	
APEX	1.22	1.10	1.00	0.88	0.81	
PLEX	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	

5. COCOMO II Post Architecture Level Equations

5.1. Effort Equation

The COCOMO II model makes its estimates of required effort (measured in Person-Months PM) based primarily on estimate of the software project's size (as measures in thousand of SLOC);

$$Effort = A * EAF * (Size)^E$$

$$A = 2.94 \text{ (for COCOMO II)}$$

$$EAF = \prod 17 CostDrivers$$

$$E = 0.91 + (\Sigma Scale\ Factors)/100$$

Where;

A = a constant that reflects a measure of the basic organizational/ technology costs.

E = an exponent derived from the five Scale factors

The Size is KSLOC. This is derived from estimating the size of software modules that will constitute the application program. It can also be estimated from unadjusted function points (UFP), converted to SLOC, then divided by one thousand.

5.2. Schedule Estimation

The initial version of COCOMO II provides a simple schedule estimation capability similar to those in COCOMO 81 and Ada COCOMO. The COCOMO II schedule equation predicts the number of months required to complete a software project. The duration of a project is based on the effort predicted by the effort equation. The initial baseline schedule equation for the COCOMO II Early Design and Post-Architecture stages is:

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

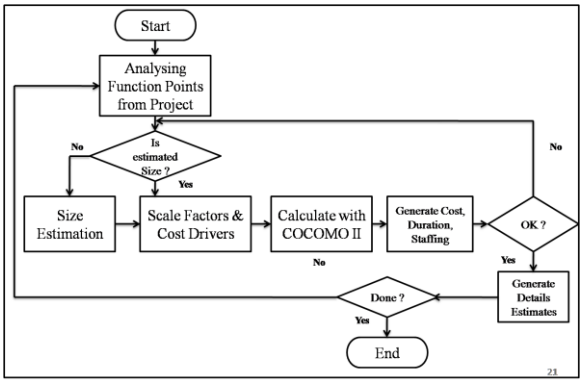
$$\mathbf{SE = 0.28 + 0.2 * (\Sigma Scale\ Factors/100)}$$

Where;

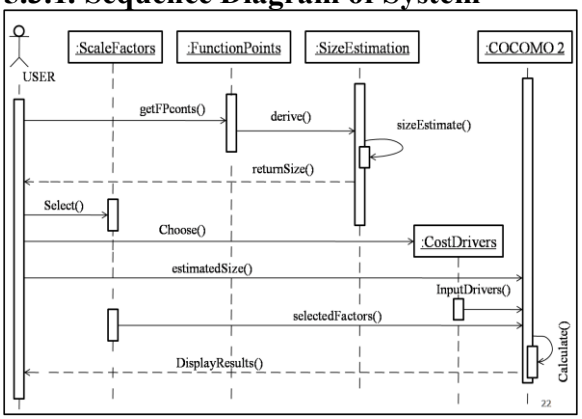
Effort = the effort from the COCOMO II effort equation

SE = the schedule equation exponent derived from the five Scale factors

5.3. System Design



5.3.1. Sequence Diagram of System



6. System Implementation

Stock Control System

As an example, a company that sells 200 different electrical goods on the phone. To do this they want you to create a computerised stock control system using JAVA language. This system should have the following functionality:

- Allow the operator to enter an existing customers number or for new customers their details (up to 100 customers)
- Check the credit rating of customers and reject those with poor ratings
- Allow the operator to enter the goods being ordered.
- Check the availability of the goods being ordered.
    - Where there are sufficient goods in stock supply all the goods
    - Where there are not sufficient goods supply number available and create back order to be supplied when goods become available.

- Update the stock levels and customer account details.
- Produce Dispatch note and invoice.
- Update stock levels based on and delivery of goods.
- Update customer account details based on payment by a customer.

Function Points Analysing Form Project

External Inputs = 6

Customer Number
New Customer Details
Order Details
Stock Delivery Details
Customer Payment Details
Main Menu Selection

External Outputs = 6

Credit Rating
Invoice
Dispatch Note
Customer Details Information
Order Details Information
Stock Details

External Inquiries = 3

Customer Details Request
Order Details Request
Stock Details Request

External Interface Files = 0

Internal Logical Files = 4

Goods Transaction File
Customer File
Goods File
Customer Transaction File

FPs Complexity Weights[3.2.1]

External Inputs	Simple	3
External Outputs	Simple	4
External inquiries	Simple	3
External Files	None	
Internal Files	Simple	7

Size Estimation[3.4]

Unadjusted FP = (6*3)+(6*4)+(3*3)+(4*7)=79
SLOC = Unadjusted FP * Language Factor
= 79 * 53 = 4187

Effort Estimation

Assuming that the project with all Nominal Scale factors and Cost drivers would have an EAF of 1.00, COCOMO II estimates that 14.20 Person-Months of effort is required to complete it:
KSLOC = SLOC / 1000 = (4187 / 1000) = 4.187

E = 0.91 + (3.72+3.04+4.24+3.29+4.68 / 100)
= 1.0997

Effort = 2.94 *(EAF) *$(Size)^E$
= 2.94 * (1.00) * $(4.187)^{1.0997}$
= 14.19887764 Person-Months

If the project is rated Very High for Complexity (effort multiplier of 1.34), and Low for Language & Tools Experience (effort multiplier of 1.09), and all of the other cost drivers are rated to be Nominal (effort multiplier of 1.00), the EAF is the product of 1.34 and 1.09.

EAF = 1.34 * 1.09 *1.00 = 1.4606

Actual Effort = 2.94 *(EAF) *$(Size)^E$
= 2.94 * (1.4606) * $(4.187)^{1.0997}$
= 20.73888067 Person-Months

Schedule and Staffing Estimation

Continuing the example, and substituting the exponent of 0.32 that is calculated from the scale factors, yields an estimate of just over a year, and an average staffing of between 2 and 3 people. Assuming Labor rate of a staff is 1000 $ per month.

SE = 0.28 + 0.2 * (ΣScale Factors/100)
= 0.28 + 0.2 * (0.1897) = 0.31794

Duration = 3.67 * $(Effort)^{SE}$
= 3.67 * $(20.73888067)^{0.31794}$
= 9.623316879 Months

Average Staffing = Effort / Duration
= 20.73888067 / 9.623316879
= 2.155065757 people

Software Cost = Effort * EAF * Labor Rate
= 20.73888067 * 1.4606 * 1000
= 30291.209 $ (in dollar)

RESULT

For Stock Control System project:
Effort = 14.20 Person-Months
Actual Effort = 20.74 Person-Months
Schedule = 9.6 Months
Average Staff = 2.16 people
Software Cost = 30292 $

8. Limitation

This System allows only software project managers and software engineers to estimate project cost and duration.

9. Conclusion

Cost estimation system shows how to make cost estimates for a project, and outlines basic steps, terms, and tools used. This system make to clarify not only an expected project cost and duration, but also prompt to verify all basic sides of software project by providing clear, compact, and concise terms, methodology, which are tested on a wide range of real-life projects, and, thus, reduce essentially project risks, and provide reasonable grounds for communication with a project. Hopefully this system delineates COCOMO II, with its possibilities, as a top contender in software cost estimation systems.

10. References

[1]. Boehm, B., et al. Software Cost Estimation with COCOMO II. Prentice Hall, 2000.
[2]. IFPUG (1994), *IFPUG Function Point Counting Practices: Manual Release 4.0*, International Function Point Users' Group, Westerville, OH.
[3]. Park R, W. Goethert, J. Webb (1994), "Software Cost and Schedule Estimating: A Process Improvement Initiative", CMU/SEI-94-SR-03, Software Engineering Institute, Pittsburgh, PA.
[4]. Abran, A. 1999. Functional Size Measurement for Real Time and Embedded Software.
[5]. Dolado, J.J. (1997) 'A study of the relationships among Albrecht and Mark II Function Points, lines of code and effort'.
[6]. Leung, H. and Fan, Z. 2002. *Software Cost Estimation, Handbook of Software Engineering*, Hong Kong Polytechnic University.
[7]. L. M. Laird, and M. C. Brennan, 2006. Software Measurement and Estimation: A Practical Approach.
[8]. Goethert, W., E. Bailey, M. Busby (1992), "Software Effort and Schedule Measurement: A Framework for Counting Staff Hours and Reporting Schedule Information." CMU/SEI-92-TR-21, Software Engineering Institute, Pittsburgh, PA