iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 4 **Issue:** VII **Month of publication:** July 2016

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Volume 4 Issue VII, July 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Estimation of Application Maintenance cost using COCOMO81

Nishu Singh[1], Anamika Mitra[2], Palvi Gupta[3]
M. Tech

Abstract: ***Constructive Cost Model is one of the CE models which contribute three different models namely basic, intermediate and detailed model. Maintenance of software is very much important so as to increase the functionality of software and decrease the cost incurred in extracting new system. The first level of COCOMO is Basic which is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes. In the Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases. Here, in this paper, we mainly estimate maintenance cost of software using basic and intermediate model by maintaining Annual change traffic and thousand source lines of code as constant. We examine these values by varying on each case, thereby obtaining the final maintenance costs in man-months.***
Keywords: CE(cost estimation), CS(Costar),

I. INTRODUCTION

The COCOMO CE model is used by thousands of software project managers, and is based on a study of hundreds of software projects. Unlike other CE models, it is an open model, so all of the details are published, including: The underlying CE equations are obtained by software maintenance cost formulated. It is well defined, and because it doesn't rely upon proprietary estimation algorithms, CS offers these advantages to its users: It is used to estimate more objective and repeatable than estimates made by methods relying on proprietary models. It can be calibrated to reflect your software development environment, and to produce more accurate estimates CS is a faithful implementation of this model that is easy to use on small projects, and yet powerful enough to plan and control large projects.

CS allows defining a software structure to meet your needs. Your initial estimate might be made on the basis of a system containing 3,000 lines of code [1]. Your second estimate might be more refined so that you now understand that your system will consist of two subsystems. There are some sizing approaches for estimating the software maintenance efforts such as source lines of code (SLOC). COCOMO is used to compute the effort and calendar time based on the size and several characteristics of the software product.

A. Review on background

COCOMO (Constructive Cost Model) is a model for estimating costs of software projects. It was created by Barry W. Boehm and published in 1981 using data collected from 63 projects. The quantity of projects studied and the care in its articulation have made the model popular and encouraged its use. The model offers empirical formulas for estimating software costs. We have taken the COCOMO as the starting point for our research, although we have borne in mind other data from the same author. After application of the first version of his model to a wide range of situations, Boehm concluded that coverage for only one mode of developing software was insufficient, so he provided for three modes: organic, semidetached and embedded. In this way, he recognized the influence of various characteristics such as size, communication needs, experience in similar projects, etc. Furthermore [2], Boehm provides the COCOMO in three versions: basic, intermediate, and detailed. The basic version is useful for quick estimations, although without fine precision. The intermediate version deals with 15 attributes of the project (reliability required, database size, memory restrictions, response time required, etc.), all of whose valuation acts as a multiplying factor in the model.

II. COCOMO MAINTENANCE CE

The detailed version deals with estimates in each of the phases in the life cycle of the project. The basic version of the model offers the following formulas to calculate the cost of software development measured in *MM* (i.e., man-months):

The first version of COCOMO is COCOMO 81. This model, extensively described *The Software Engineering Economics* is still an interesting model to understand. The COCOMO 81 model is a static model with a hierarchy of three estimation models [3].

A. Basic Model

It provides a rough estimate of effort based on size and mode of software development.

B. Intermediate Model

It refines the basic model by use of 14 cost drivers-these are subjective attributes. The impact of cost drivers is considered at the project level, Effort is further adjusted to take into account any schedule constraint.

C. Detailed Model

This model further refines the estimation by considering the phase-wise impact of cost-drivers, and the computations in this are for various sub-systems/modules. All the three COCOMO 81 models use size as the main input. Another factor used in all three models is the "mode" of project development- the way the project is going to be handled. The three modes considered are:

1) Organic Mode: Project developed by a small, experienced team working in a familiar environment. The team communicates easily. Projects done like these are small or medium sizes with minimal need for innovations. Environment, hardware and tools are stable.

2) Semi-detached Mode: This mode lies between organic and embedded mode.

3) Embedded Mode: Projects developed by large teams where communication is not achieved easily. Projects are medium to large, being developed under tight constraints, possibly with instable hardware and software environment and requiring complex, innovate processing. **A) Basic Model: [3]** The Basic Model is the simplest model and does not consider any cost drivers, using the size as the basis for estimation. It is approximately only for a rough and ready estimate. The following is the calibration for size specified in KLOC. This is based on project data collected from multiple sources. Organic mode: *MM*DEV = 2·4 *KS*1·05 (1) Semidetached mode: *MM*DEV = 3·0 *KS*1·12 (2) Embedded mode: *MM*DEV = 3·6 *KS*1·20 (3) Where *KS* is an estimate of the delivered program size in thousands of instructions (or roughly, lines of source code). To estimate the maintenance cost, another parameter is needed: the annual change traffic (*ACT*) which consists of the proportion of original instructions that undergo a change during a year by addition or modification. *ACT* = (*NNL* + *NML)/(NOL)* (4) generators is excluded. The original COCOMO 81 model was defined in terms of Delivered Source Instructions, which are very similar to SLOC. The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI. Glance of CM: The COCOMO CE model is used by thousands of software project managers, and is based on a study of hundreds of software projects. Unlike other CE models, it is an open model, so all of the details are published, including: The underlying CE equations are obtained by software maintenance cost formulated. It is well defined, and because it doesn't rely upon proprietary estimation algorithms, CS offers these advantages to its users: Its estimates are more objective and repeatable than estimates made by methods relying on proprietary models it can be calibrated to reflect your software development environment, and to produce more accurate estimates CS is a faithful implementation of this model that is easy to use on small projects, and yet powerful enough to plan and control large projects. Typically, you'll start with only a rough description of the software system that you'll be developing, and you'll use CS to give you early estimates about the proper schedule and staffing levels. As you refine your knowledge of the problem, and as you design more of the system, you can use CS to produce more and more refined estimates. CS allows defining a software structure to meet your needs. Your initial estimate might be made on the basis of a system containing 3,000 lines of code. Your second estimate might be more refined so that you now understand that your system will consist of two subsystems. There are some sizing approaches for estimating the software maintenance efforts such as source lines of code (SLOC).

III. COST DRIVERS

COCOMO II has 17 cost drivers to assess project, development environment, and team to set each cost driver. The cost drivers are multiplicative factors that determine the effort required to complete your software project. For example, if a project will develop software that controls an airplane's flight, you would set the Required Software Reliability cost driver to Very High. That rating corresponds to an effort multiplier of 1.26, meaning that your project will require 26% more effort than a typical software project. Constructive Cost Model is used for estimating costs of software projects. It was created by Barry W. Boehm and published in 1981 using data collected from 63 projects. The quantity of projects studied and the care in its articulation have made the model popular and encouraged its use. The model offers empirical formulas for estimating software costs. We have taken the COCOMO as the starting point for our research. After application of the first version of his model to a wide range of situations, Boehm concluded that coverage for only one mode of developing software was insufficient, so he provided for three modes: organic, semidetached and

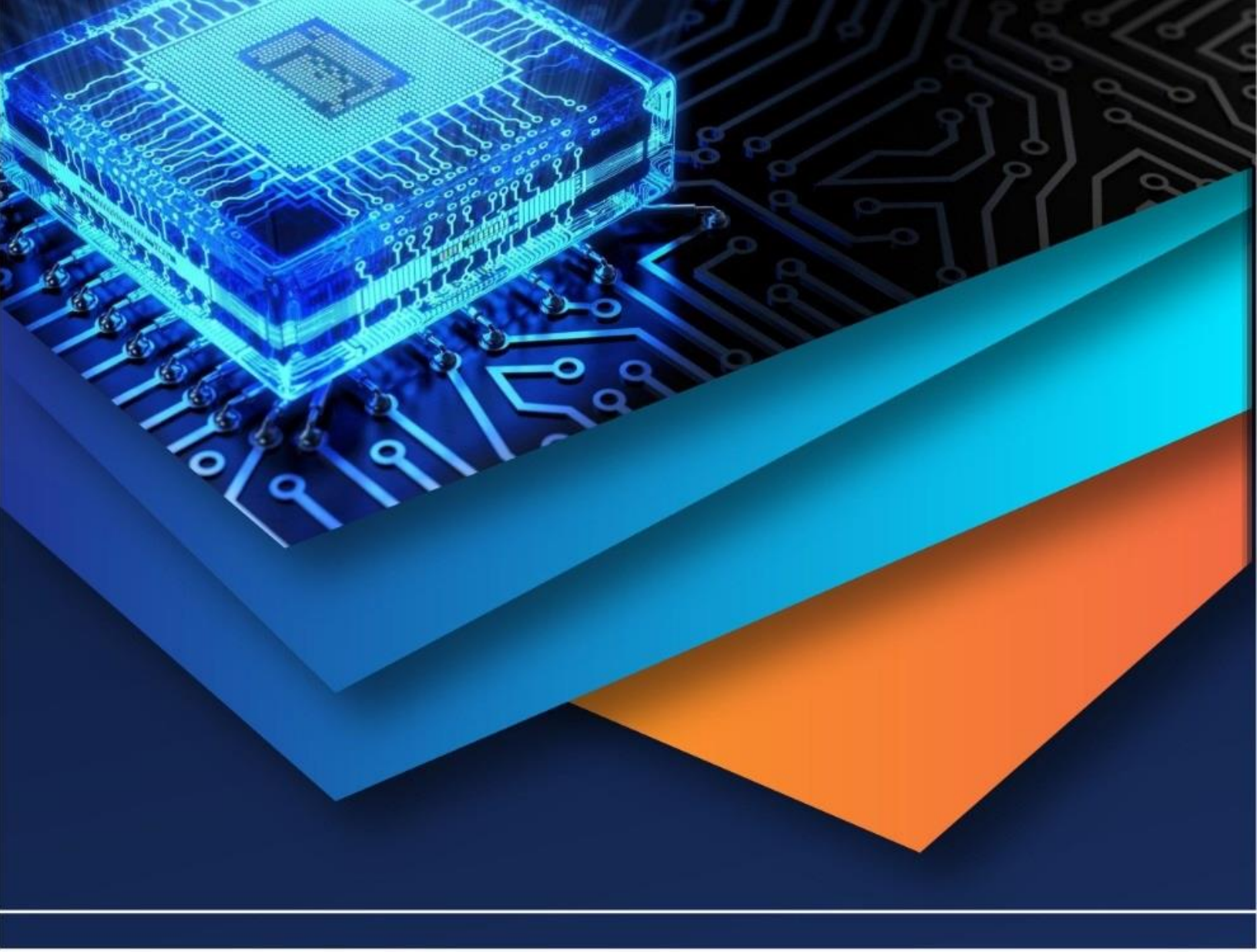
embedded. In this way, he recognized the influence of various characteristics such as size, communication needs, experience in similar projects, etc. Furthermore, Boehm provides the COCOMO in three versions: basic, intermediate, and detailed. The basic version is useful for quick estimations, although without fine precision. The intermediate version deals with 15 attributes of the project (reliability required, database size, memory restrictions, response time required, etc.), all of whose valuation acts as a multiplying factor in the model.

IV. CONCLUSION

Annual change traffic as well as Lines of code plays a crucial role in finding out the differences in new results obtained in basic and intermediate model. The main aim in taking up this paper is to compare and evolve the changes occurred when new set of resulted values are formed. This comparison helps the software engineering professionals to know the cost decreasing principles in software maintenance. Finally, we conclude this paper by obtaining the final maintenance costs and cost decreasing factors. As part of the future work, this project can be extended by assigning random multiplier values and check the difference in variations of software maintenance. The global aim here is to decrease maintenance costs at maximum extent so as to encourage the software developers to go for maintaining of software at the time of malfunctioning instead of going for new product, thereby benefiting the financial status of Information Technology sector in India.

REFERENCES

[1] "An Approach to Find Maintenance Costs Using Cost Drivers of Cocomo Intermediate Model" Vol 3 Issue. 1, January 2013, pg.154-158 in International Journal of Computational Engineering Research by cvsr syavasya
[2] http://en.wikipedia.org/wiki/COCOMO
[3] "Software Requirements and Estimation" by Swapna Kishore and Rajesh Naik published by Tata McGraw Hill.
[4] http://www.softstarsystems.com/overview.htm



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY