Comparative Analysis for Traditional and Object- Oriented Approach using COCOMO-II Model for Cost Estimation

[1] Amar Pal Yadav **[2] C. S. Yadav** **[3] Ram Kumar Sharma**

Abstract: The role to measuring the software development effort for the line of code, use case point effort and development effort using different modules. The problem of cost estimation in software engineering has been addressed by several researchers. The aim of this research is to identify the use of complexity metrics as important parameters of programming language based on different attributes for effort estimation. There are many software metrics used for cost estimation. In this paper, introduce new approach with COCOMO II model for estimating the cost based on LOC metrics for different traditional and object oriented programming languages. Also, it describes software metrics used for software project cost estimation. This paper summarizes existing literature on software project cost estimation.

Keywords: Lines of code, Performance Evaluation, Traditional Approach, Object-Oriented Approach, Analysis, Design, Deployment, Test, Methodology, Metric, Complexity, development effort, software measurement, effort estimation, COCOMO II.

I- INTRODUCTION

There are a number of competing software cost estimation methods available for software developers to predict effort and cost. Software metrics gather information about a product based on its measurable characteristics and hence provide good criteria for measuring the similarity of several software products. In this paper, a survey of several cost estimation related software metrics is carried out.

Manuscript received May 2015

AMAR PAL YADAV *M.Tech. UPTU / Noida Institute of Engineering and Technology Gr. Noida. Mob-9953148371 India.*

C. S. YADAV *Associate Professor and Head, Computer Science & Engineering Department at Noida Institute of Engineering & Technology (NIET), Greater Noida India.*
Mob No-09313419956

The goal is to evaluate these metrics and provide a common standard in terms of software metrics and attributes that are related to cost estimation [2, 4]. Software cost estimation of a project is pivotal to the acceptance or rejection of the development of software project. Various Software cost estimation techniques have been in practice with their own strengths and limitations. The latest of these is object-oriented one. Currently object-oriented approach for Software cost estimation is based on Line of Code (LOC), function points, functions and classes etc. This paper summarizes research in deriving a baseline COCOMO 2.0 model tailored to these new forms of software c, including cost estimation rationales for the model decisions [1, 3].

II- PROBLEM STATEMENT

Traditional software size metrics analysis depend upon Line of code, Function point analysis, Software size, Extension of function point, Object point etc are used. Among all the methods Line of Code and function point are the most popular metrics [5]. But both metrics along with their limitations are as follows:

A. Line of Code:

The Line of code is the oldest, simplest and most widely used metrics for calculation of program size. It counts the 'Number of Instructions' of a program excluding comments and blank lines in terms of SLOC (Source Lines of Code). SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements [6].

RAM KUMAR SHARMA: *Department of Computer Science & Engineering, Noida Institute of Engineering and Technology Greater Noida Mob No-09654624322*

ISSN: 2278 – 1323

B. Function Point:

Function Point analysis is one of the best methods in traditional metrics for measuring the size of software.

The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Function points represent logical size, as opposed to physical size (like SLOC or objects) [7].

So Object-Oriented Measurement has become an increasingly popular research area. Metrics are the powerful support tools in software development and maintenance. These are use to access software quality, to estimate complexity, cost and effort, to control and improve processes. The metrics that is important to calculate effort between Traditional and Object-Oriented approach [8, 9].

TRADITIONAL VS. OBJECT ORIENTED METRIC:

The recent trend of using Object Oriented tends developer estimating the size of their development projects. Traditional software measurement techniques have proven unsatisfactory for measuring productivity and predicting effort. There are many aspects that an Object Oriented metric must have if it has to provide accurate effort prediction. Also, it is important to include information about communication between objects and reuse through inheritance in the 'size' as well. Unlike traditional metrics, which are based on the data and procedure model of structured analysis, Object Oriented metrics are based on the objects and their characteristics [6]. The limitations of traditional methods, when applied to Object Oriented solution are that, they tend to measure only one aspect of the software i.e. the functionality. Functionality is required when effort is need to be predicted. However, considering only this aspect, particularly in a well-designed OO solution is not sufficient, as in addition to functionality, a level of complexity is also added to the software that depends on the amount of communication between the objects in the system. Another important aspect of object-oriented size is reuse through inheritance. A good object-oriented analysis involves identifying groups of objects (actors) whose behaviors are similar, number of classes and number of methods [10]. The primary objectives for Object Oriented metrics are no different than those for metrics derived for conventional software and aims at:

• To better understand the quality of the product

• To assess the effectiveness of the process

• To improve the quality of work performed at a project level various object oriented metrics have been proposed in literature [11].

Object-Oriented Measurement has become an increasingly popular research area. Metrics are the powerful support tools in software development and maintenance. These are use to access software quality, to estimate complexity, cost and effort, to control and improve processes. The metrics that is important to calculate effort between Traditional and Object-Oriented approach [12].

(**i.**) CYCLOMATIC COMPLEXITY (CC):

Cyclomatic complexity is a measure of a module that control flow complexity based on graph theory [5]. Cyclomatic complexity of a module uses control structures to create a control flow matrix, which is used to generate a connected graph. These graphs represent the control paths through the module. The complexity of the graph is the complexity of the module. Fundamentally, the CC of a module is roughly equivalent to the number of decision points and is a measure of the minimum number of test cases that would be required to cover all execution paths. A high Cyclomatic complexity indicates that the code may be of low quality and difficult to test and maintain [6].

(**ii**) SOURCE LINES OF CODE (SLOC):

The SLOC metric measures the number of physical lines of active code, that is, no blank or commented lines code [10]. Counting the SLOC is one of the earliest and easiest approaches to measuring complexity. In general the higher the SLOC in a module the less understandable and maintainable [13].

ISSN: 2278 – 1323

(**iii**) COMMENT PERCENTAGE (CP):

The CP metric is defined as the number of commented lines of code divided by the number of non-blank lines of code. Usually 20% indicates adequate commenting for C++. A high CP value facilitates in maintaining a system [11].

III- PROPOSED WORK AND METHODOLOGY

Presents an overview of the metrics applied for object oriented systems. The new approach supports the continued use of traditional metrics, but within the structures and confines of object oriented systems [13, 14].

The first two metrics in Table 1 are examples of traditional metrics applied to the object oriented structure of methods instead of functions or procedures. The next six metrics are specifically for object oriented systems and the object oriented construct applicable is indicated.

TABLE: 1

SOURCE	**METRIC**	**TRADITIONAL/OO**
TRADITIONAL	CC	METHOD
TRADITIONAL/OO	LOC	METHOD
OO	RFC	CLASS/METHOD
OO	WMC	CLASS/METHOD
OO	LCOM	CLASS/COHESION
OO	DIT	INHERITENCE
OO	NOC	COUPLING

Figure: 1 Metrics for OO and Traditional system

In Table 1used for calculating and determining the project cost on the source code length with different attribute parameters like as methods, classes, inheritance etc. There is various type of metric for estimating the cost of traditional and object-oriented, but in our approach used only LOC metrics for comparative analysis on different attributes.

We implemented with common approach traditional and object oriented by using the LOC metrics.

(i) Metric measurement Line of Code (LOC): In this measurement we obtain a count of total number of lines in the module. It includes source lines, blank lines, comment lines.

a- Physical Line of Code:

This measure provides a count of total number of source line in the module.

b- Number of Statement:

This measure indicates total number of statement in the module. It includes if, else, switch case, while, do while, for statements.

c- Comment Lines:

This measure indicates total number of comment lines in a module.

d- Blank Line:

This measure indicates total number of blank lines in a module.

e- Non-comment and non-blank (NCNB):

This measure provides count of all lines that are not comments and not blanks.

(ii) Project wide Metrics:

a- Reuse Ratio (R):

The reuse ratio (R) is given by R=Number of super class/Total number of class.

b- Specialization Ration (S):

This ratio measures the extent to which a super class has captured abstraction.

S=Number of subclass/ number of super class.

c- Average Inheritance Depth

The cost estimation model used in the calibration is COCOMOII early design model. The estimation of the project size will be in Object Oriented Function Points (OOFP) instead of traditional Function point (FP) used in COCOMOII model [16].

ISSN: 2278 – 1323

Estimate the effort required to finish this project using calibrated COCOMOII early design model

Effort PM= A * (Size) B * EM1* EM2*……* EM7

A: is the constant (calibrated according to historical data of the information)

Size: the estimated thousand Line of Code (KLOC)

B: Exponent value calculated for each project by considering five scale factors described in COCOMOII.

EM: is the effort multiplier according to COCOMOII values

i. Estimate the cost of the project in SR by the equation

ii. Estimated Software Cost = Estimated Effort * Labor Rate of the organization.

IV- RESULTS AND DISCUSSION

Object Oriented Metrics in Software Engineering Approach "Given the central role that software development plays in the delivery and application of Information Technology, managers are increasingly focusing on process improvement in the software development area. This demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO) [15]. In addition, the focus on process improvement has increased the demand for software measures, or metrics".

The Software cost estimation techniques provides the support for minimal line of code in a project. There are different techniques for cost estimation, but the object oriented model using COCOMO II is the best model for calculating the cost.

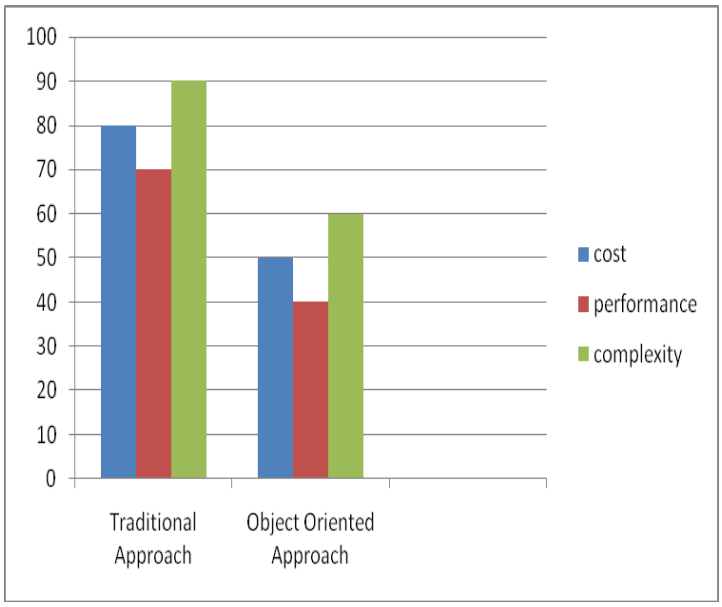

Figure 2: Illustrate the Different Criteria (Complexity, Performance and Cost) for Traditional Approach and Object-oriented Approach.

In Traditional Approach this criterion depends on the type of model and size of project, but in general as shows from figure1 is little above from the middle, however the Object-Oriented Approach depends on the complexity of project that leads to increase the cost than other approach. The traditional approach uses traditional projects that used in development of their procedural programming like C; this approach leads software developers to focus on Decomposition of larger algorithms into smaller ones. The object-oriented approach uses to development the object-oriented projects that use the object-oriented programming like: C++ and Java.

The object-oriented approach to software development has a decided advantage over the traditional approach in dealing with complexity and the fact that most contemporary languages and tools are object-oriented.

V- CONCLUSION & FUTURE WORK

A metric for software model complexity which is a combination of some of the metrics mentioned above with a new approach. With this metric we can measure software's overall complexity. Traditional Methods used for size

ISSN: 2278 – 1323

estimation requires lot of efforts and do not give accurate results. Also they do not support the features of newer and rapid technologies like Object Oriented technologies.

In an Object Oriented Paradigm, several metrics are suggested by various researchers for size estimation. We estimate the cost of software project using COCOMOII model based on LOC. By using the COCOMOII model metrics gives accurate results with less effort than traditional methods. Also these metrics take care of complexity, which is a crucial aspect for size estimation. Size being an important factor for the effort/cost/duration estimation. However, manual efforts are required for estimating the same. Further studies are required for proposing new metrics which can automate the process of size estimation. In future we can also work various other metrics for cost estimation on RFC, WMC etc.

REFRENCES

[1] An inheritance complexity metric for object-oriented code:A cognitive approach SANJAY MISRA, IBRAHIM AKMAN and MURAT KOYUNCU vol-26 part 3 june 2011

[2] Patrick Naughton & Herbert Schildt "java:The complete reference", McGraw-Hill Professional, UK, 2008.

[3]Er. V.K. Jain. "The Complete Guide to java programming", First Edition, 2001.

[4]Mike O'Docherty, "Object-Oriented Analysis and Design Understanding System Development with UML 2.0", John Wiley & Sons Ltd, England, 2005.

[5] Magnus Christerson and Larry L. Constantine, "Object-Oriented Software Engineering- A Use Case Driven Approach ", Objective Systems, Sweden, 2009.

[6]V. Krishnapriya, Dr. K. Ramar, "Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures", 2010 International Conference on Advances in Computer Engineering, 978-0-7695-4058-0/10 $26.00 © 2010 IEEE.

[7] McCabe, T. J., "A Complexity Measure", IEEE Transactions on Software Engineering, SE 2(4), pages 308-320, December 1976.

[8] Selim Kebir, Abdelhak-DjamelSeriai, Sylvain Chardigny and Allaoua Chaoui "Quality-Centric Approach for Software Component Identification from Object-Oriented Code"IEEE 2012 Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture page no. 181-190.

[9]Shyam R. Chidamber, Chris F. Kemerer, A METRICS SUITE FOR OBJECT ORIENTED DESIGN, 1993

[10]Carnegie Mellon School of Computer Science, Object-Oriented Testing & Technical Metrics, PowerPoint Presentation, 2000

[11] Sencer Sultanoðlu, Ümit Karakaþ, Object Oriented Metrics, Web Document, 1998

[12] Linda H. Rosenberg, Applying and Interpreting Object Oriented Metrics Sencer Sultanoðlu, Ümit Karakaþ, Complexity Metrics and Models, Web Document, 1998

[13] Boehm, Barry W., as quoted by Ware Myers, "Software Pivotal to Strategic Defense," IEEE Computer, January 2001.

[14] Campbell, Luke and Brian Koster, "Software Metrics: Adding Engineering Rigor to a Currently Ephemeral Process," briefing presented to the McGrummwell F/A-24 CDR course, 2003.

[15] V. Basili, Qualitative Software Complexity Models: a Summary, in Tutorial on Models and Methods for Software Management and Engineering, IEEE Computer Society Press, Los Alamitos, CA, 2004.

[16]E. Weyuker, Evaluating Software Complexity Measures, IEEE Trans. Software Eng., 14(9), 2002, pp. 1357-1365.

ISSN: 2278 – 1323

ABOUT AUTHORS:

1. 

AMAR PAL YADAV: M.TECH. Department of Computer Science & Engineering, Noida Institute of Engineering and Technology Greater Noida.

Mob No-09953148371

2. 

C. S. YADAV: Prof. Chandra Shekhar Yadav is an Associate Professor and Head, Computer Science & Engineering Department at Noida Institute of Engineering & Technology (NIET), Greater Noida. He has about 17 years of experience in teaching. He received Master of Computer Application degree from Institute of Engineering & Technology (IET), Lucknow in year 1998, M. Tech (Computer Science & Engineering) from JSSATE, Noida in year 2007. Currently he has submitted Ph.D. (Computer Science & Engineering) thesis in U.P. Technical University, Lucknow under the supervision of Prof. (Dr.) Raghuraj Singh, Director KNIT Sultanpur. He has supervised 07 M. Tech. theses.

Mob No-09313419956

3. 

RAM KUMAR SHARMA: Department of Computer Science & Engineering, Noida Institute of Engineering and Technology Greater Noida.

Mob No-09654624322

ISSN: 2278 – 1323