Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

Software Cost Estimation using Function Point with Non Algorithmic Approach

V.Vignaraj Ananth[1], Dr. N. Balaji[2] and N.Shivakumar[3]

[1] Thiagarajar College of Engineering

Received: 14 December 2012 Accepted: 2 January 2013 Published: 15 January 2013

Abstract

Cost estimation is one of the most challenging tasks in project management. It is to accurately estimate needed resources and required schedules for software development projects. The software estimation process includes estimating the size of the software product to be produced, estimating the effort required, developing preliminary project schedules, and finally, estimating overall cost of the project. Nearly one-third projects over run their budget and late delivered and two-thirds of all major projects substantially over run their original estimates. Effort is a function of size. For estimating effort first we face sizing problem. In direct approach size is measured in lines of code (LOC). In indirect approach, size is represented as Function Points (FP). In this paper we use both approach with different technique.

Index terms— estimation; budget; effort; LOC; FP.

1 Introduction

ut of the three principal components of cost i.e., hardware costs, travel and training costs, and effort costs, the effort cost is dominant. Software cost estimation starts at the proposal state and continues throughout the life time of a project.

There are several techniques of software cost estimation:

? Algorithm Cost Model ? Expert Judgments ? Estimation by Analogy ? Top-down Estimation ? Bottom-up Estimation a) Expert Judgment Method Expert judgment techniques involve consulting with software cost estimation expert or a group of the experts to use their experience and understanding of the proposed project to arrive at an estimate of its cost.

2 b) Estimating by Analogy

Estimating by analogy means comparing the proposed project to previously completed similar project where the project development information id known. Actual data from the completed projects are extrapolated to estimate the proposed project. This Author ? : Professor and Head Information Technology KLN College of Engineering Madurai, India. E-mail : balajin@klnce.edu Author ? : Assistant Professor, Department of Computer Science Thiagarajar College of Engineering Madurai, India. E-mail : shiva@tce.edu Author ? : PG Student, Department of Computer Science Thiagarajar College of Engineering Madurai, India. E-mail : vignarajcse@tce.edu method can be used either at system-level or at the component-level.

3 c) Top Down Estimating Method

Top-down estimating method is also called Macro Model. Using top-down estimating method, an overall cost estimation for the project is derived from the global properties of the software project, and then the project is partitioned into various low-level components.

4 d) Bottom Up Estimating Method

Using bottom-up estimating method, the cost of each software components is estimated and then combine the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge accumulated about the small software components and their interactions.

5 e) Algorithmic Method

The algorithmic method is designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and historical data and use inputs such as Source Lines of Code (SLOC), number of functions to perform, and other cost drivers.

6 II.

7 Direct Approach

Source lines of code (SLOC) is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced. There are two major types of SLOC measures: physical SLOC (LOC) and logical SLOC (LLOC). Specific definitions of these two measures vary, but the most common definition of physical SLOC is a count of lines in the text of the program's source code including comment lines. Blank lines are also included unless the lines of code in a section consists of more than 25% blank lines. Logical SLOC attempts to measure the number of executable "statements", but their specific definitions are tied to specific computer languages.

The COCOMO cost estimation model is used by thousands of software project managers, and is based on a study of hundreds of software projects. Unlike other cost estimation models, COCOMO is an open model. COCOMO estimates are more objective and repeatable than estimates made by methods relying on proprietary models. The most fundamental calculation in the COCOMO model is the use of the Effort Equation to estimate the number of Person-Months required to develop a project. COCOMO has cost drivers that assess the 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. number of executable "statements", but their specific definitions are tied to specific computer languages. Effort is calculated by Effort= a* Where 'a' and 'b' are empirically determined constants. Size is length of the code in KLOC.

The Effort Adjustment Factor in the effort equation is simply the product of the effort multipliers corresponding to each of the cost drivers.

For example, if your 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.

The COCOMO schedule equation predicts the number of months required to complete your software project. The duration of a project is based on the effort predicted by the effort equation: LOC and FP. Programming language levels and average numbers of source code statements per function point.

8 d) Fuzzy Logic

Fuzzy logic is used to find fuzzy functional points and then the result is defuzzified to get the functional points and hence the size estimation in person hours. Triangular fuzzy numbers are used to represent the linguistic terms in Function Point Analysis (FPA) complexity matrixes. A fuzzy set is characterized by a membership function, which associates with each point in the fuzzy set a real number in the interval [0,1], called degree or grade of membership. The membership function may be triangular, trapezoidal, parabolic etc. Fuzzy numbers are special convex and normal fuzzy sets, usually with single modal value, representing uncertain quantitative information. A triangular fuzzy number (TFN) is described by a triplet (?, m, ?), where m is the modal value, ? and ? are the right and left boundary respectively.

We take each linguistic variables as a triangular Fuzzy numbers, TFN (?, m, ?), ? ? m, ? ? m. The membership function (?(x)) for which is defined as:

The five major components mentioned above, they have to be rated as either Low, Average, or High. Ranking is commonly based on File Types Referenced, Data Element Types and Record Element Types. File Types Referenced (FTRs) represents the total number of internal logical files (ILFs) maintained, read, or referenced and the external interface files read or referenced by the EI/EO transaction. Data Element Type (DET) can be defined as unique user recognizable nonrecursive fields including foreign key attributes that are maintained on ILF/EIF. Record element type (RET) is a subgroup of data elements within an ILF/EIF. For each of the components belonging to Transactional functions, the ranking is based on the number of files updated or referenced (FTRs) and number of data element types (DETs). For the data components viz., Internal Logical Files (ILF) and External Interface Files (EIF), ranking is based on the number of Data Element Types (DETs) and number of Record Element Types (RETs). Based on the ratings the domain character values are fuzzified using the Triangular membership function. The value thus obtained is called membership function output, whose domain is specified, usually the set of real numbers and whose range is the span of positive numbers in the closed interval

[0, 1]. Each numerical value of the domain is assigned a specific value and 0 represents the smallest possible value of the membership function, while the largest possible value is 1.

9 e) Defuzzification

Defuzzification means the fuzzy to crisp conversions. The fuzzy results generated cannot be used as such to the hence it is necessary to convert the fuzzy quantities into crisp quantities for further processing. This can be achieved by using defuzzification process. The defuzzification has the capability to reduce a fuzzy to a crisp single-valued quantity or as a set, or converting to the form in which fuzzy quantity is present. Defuzzification can also be called as "rounding off" method. Defuzzification reduces the collection of membership function values in to a single sealer quantity.

Defuzzification is the process of producing a quantifiable result in fuzzy logic, given fuzzy sets and corresponding membership degrees. It will have a number of rules that transform a number of variables into a fuzzy result, that is, the result is described in terms of membership in fuzzy sets. The defuzzification is applied to the value that had been obtained from the fuzzification process. The fuzzified output has to be defuzzified into the real number so that it will give the effort that has been needed for the cost estimation.

10 IV.

11 Various Criterions for Assessment of Software Cost Estimation Models

There are 4 important criterions for assessment of software cost estimation models:

12 ?ð ??"ð ??"(?? ?ð ??"ð ??"

) *

V.

13 Experimental Results

Performance of the effort can be predicted based on the MARE and Prediction n method. The estimated effort of LOC is compared with the actual effort of LOC in the first graph. The estimated effort of FP is compared with the actual effort of FP in the second graph. The MARE of LOC and FP is compared in the third graph. It has been clearly identified that Function point based estimation is better than the LOC estimation.

The Table **??** indicates the lines of code with the actual effort and the estimated effort using the cocomo model. Both MARE analysis and Prediction n method has been applied to the direct approach and the indirect approach. The actual effort is the original effort and the estimated effort is the one which has been done in the estimation process using the cocomo method.

The next table shows the function point with actual effort and the estimate effort.

The graph shows the variation between the actual and estimated effort using LOC.

14 LOC

15 Conclusion and Future Work

This project proposes an efficient way of estimating the effort. The results of the estimation based on the Direct method shows that the deviation between the actual and the estimated effort is more. The result of Indirect method using the algorithmic technique cocomo model based estimation reduces the relative error and the mean absolute relative error. So the analysis of the effort from Direct method and Indirect method gives that Function point based estimation is the efficient method for the estimation process.

Though Cocomo model which is algorithmic method is an open model. It has some limitations also. In the FP based estimation also exists the deviation between actual and estimated effort. So the same effort can be implemented by using the Non algorithmic Method. Fuzzy logic is one type of Non algorithmic method. This fuzzy based estimation using the Triangular Membership Function has been proposed in this paper. In future this non algorithmic based estimation can be done to achieve the better performance.

2013 

Figure 1: O © 2013

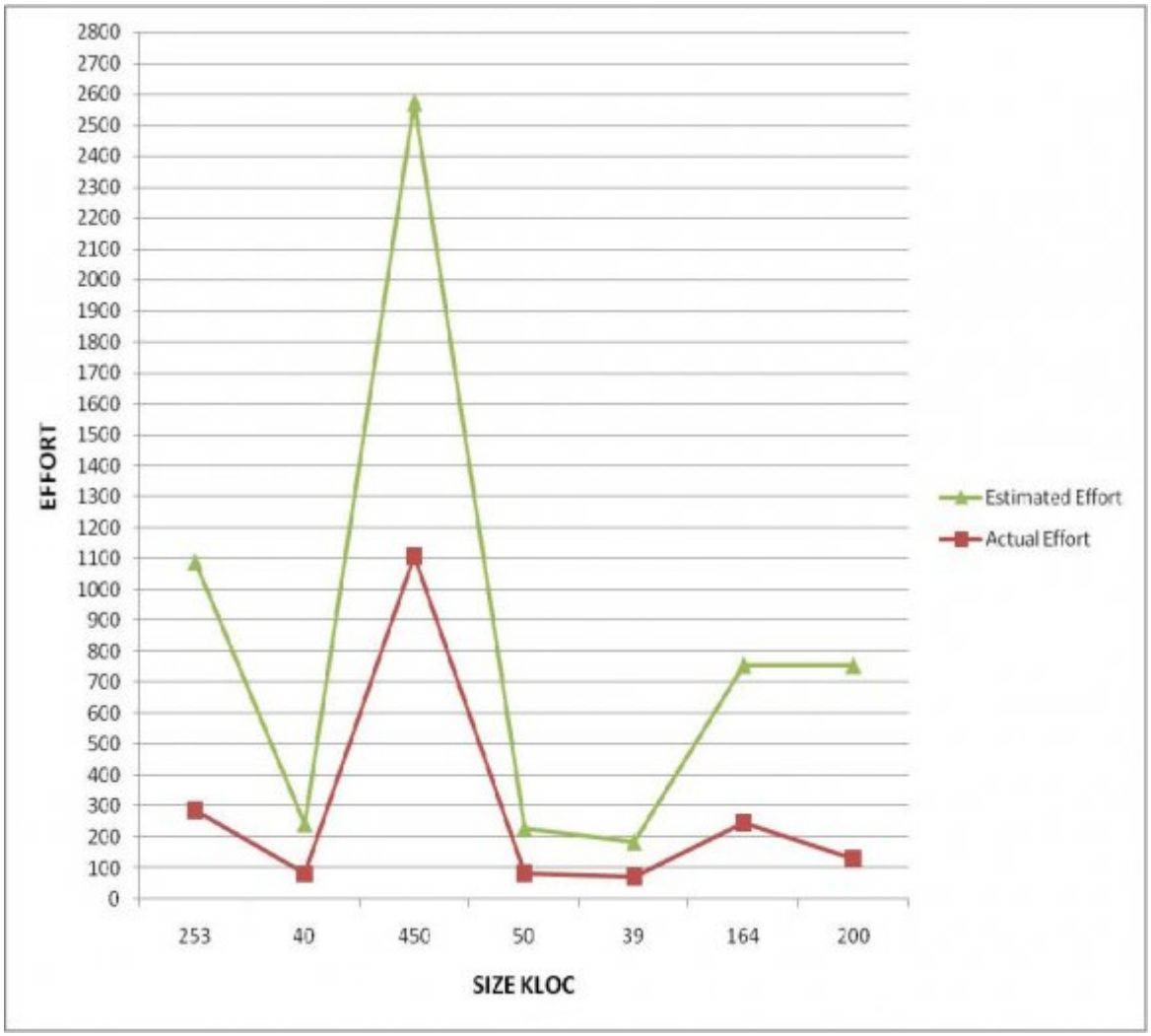
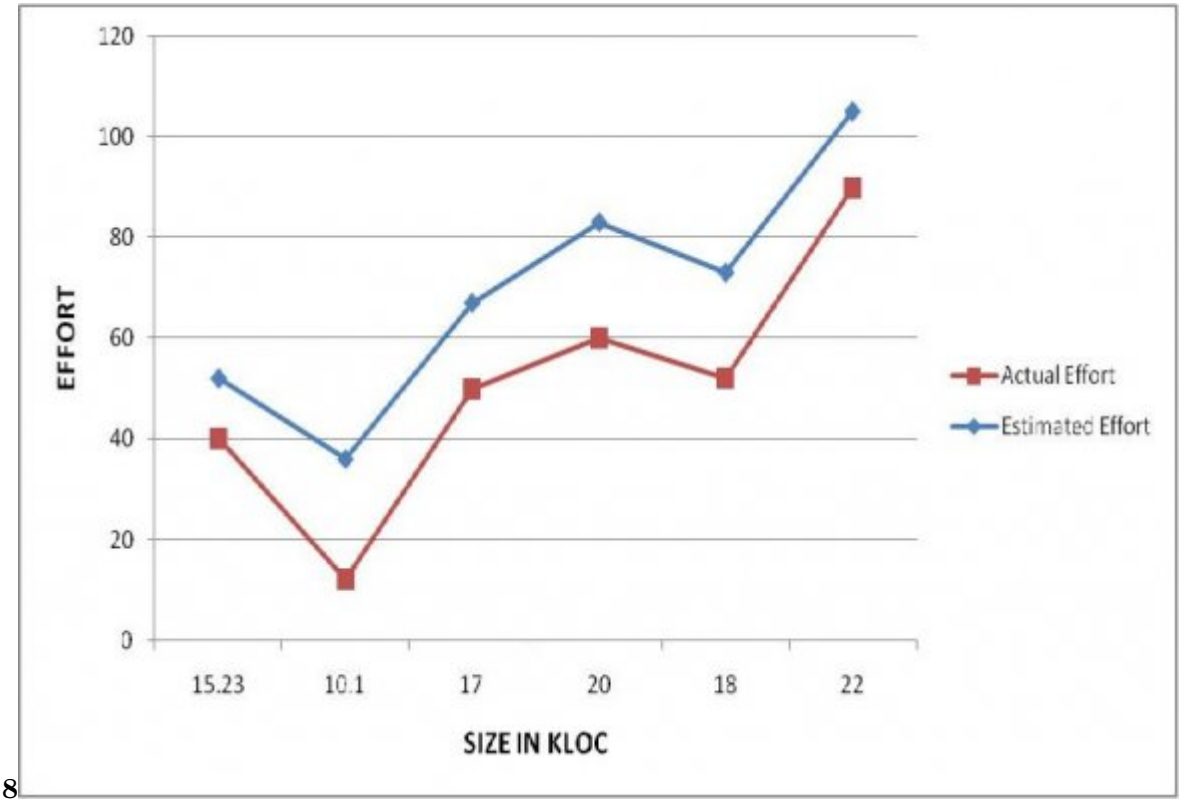

EFFORT
2800
2700
2600
2500
2400
2300
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
253
40
450
50
39
164
200
SIZE KLOC
Estimated Effort
Actual Effort

Figure 2:



18

Figure 3: 1 .C 8 4.

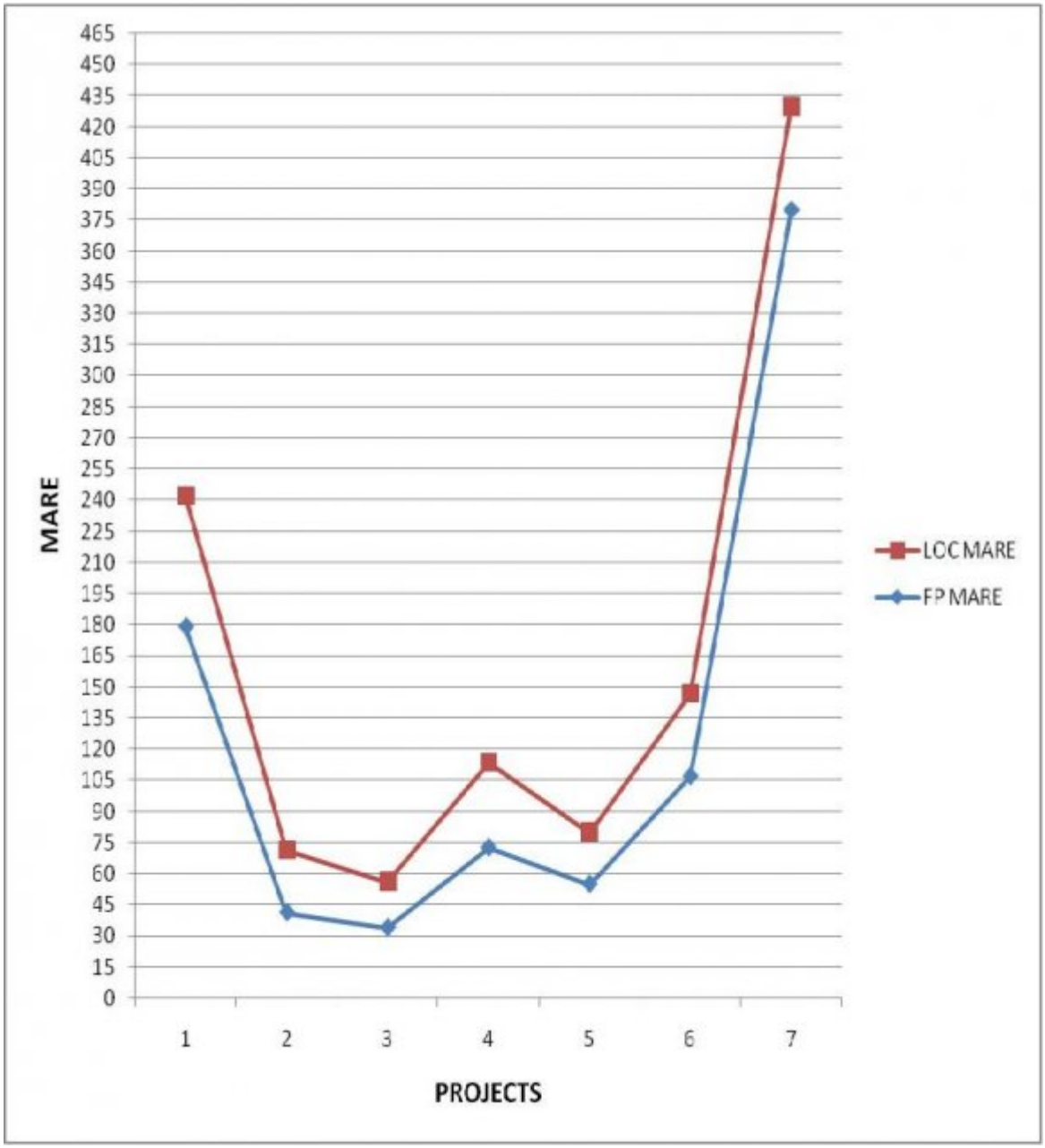

MARE
465
450
435
420
405
390
375
360
345
330
315
300
285
270
255
240
225
210
195
180
165
150
135
120
105
90
75
60
45
30
15
0
1
2
3
4
5
6
7
PROJECTS
LOC MARE
FP MARE

Figure 4: C

[Baiely and Basili ()] 'A Meta model for Software Development Resource Expenditure'. J Baiely , Basili . *Proc. Intl. Conference Software Egg*, (Intl. Conference Software Egg) 1981. p. .

[Benediktsson et al. ()] 'COCOMO based effort estimation for iterative and incremental software development'. O Benediktsson , D Dalcher , K Reed , M Woodman . *Software Quality Journal* 2003. 11 p. .

[Idri and Kijri (2000)] 'COCOMO cost modeling using Fuzzy Logic'. Ali Idri , Abran , Laila Kijri . *International conference on Fuzzy Theory and technology Atlantic, 7 New Jersy*, March 2000.

[Sheta ()] 'Estimation of the COCOMO Model Parameters Using Genetic Algorithm for NASA Software Projects'. Alaa F Sheta . *Journal of Computer Science* 2006. 2 (2) p. .

[Ifpug ()] *Function Point Counting Practices Manual: Release 4.0. International Function Point Users Group*, Ifpug . 1994. Princeton Junction, NJ.

[Idri and Abran] A Idri , A Abran . *COCOMO Cost Model Using fuzzylogic*,

[Boehm et al. (2000)] *Software cost estimation with COCOMO II*, B Boehm , C Abts , A W Brown , S Chulani , B K Clark , E Horowitz , R Madachy , D J Reifer , B Steece . February 2000. Upper Saddle River, NJ: Prentice-Hall.

[Boraso et al. ()] *Software cost estimation: An experimental study of model performances*, M Boraso , C Montangero , H Sedehi . 1996. (tech. rep.)

[Menzies et al. ()] 'Validation Methods for calibrating software e_ort models'. T Menzies , D Port , Z Chen , J Hihn , S Stukes . *ICSE '05: Proceedings of the 27 th international conference on Software engineering*, (New York, NY, USA) 2005. ACM Press. p. .