ISSN 2348 – 8034
Impact Factor- 5.070

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

SOFTWARE COST ESTIMATION USING MODEL BASED TECHNIQUES: AN OVERVIEW

Javaid Iqbal Bhat*[1] & Zahid Hussain Wani[2]
*[1]Assistant Professor, Department of Computer Science, Islamic University of Science & Technology, Kashmir, INDIA
[2]Assistant Professor, Department of Computer Science, Islamic University of Science & Technology, Kashmir, INDIA

ABSTRACT

Software cost estimation is the forecasting of development effort and development time needed to develop a software project. It is considered to be the primary step of software development process and at the same time considered to be the key task as accurate assessments of growth of the current project, its delivery exactness and its cost control can only be achieved once calculated estimation is accurate. And at broader perspective an accurate estimation of a currently developing software project will result in landing the organization in a better schedule of its futuristic software projects too. With due above reason, software effort estimation has received a considerable amount of attention of many researchers for past so many decades. And accordingly, a good number of software cost estimation techniques from last so many decades have been proposed which differ from model based techniques, also called as algorithmic techniques to Non algorithmic to Data mining and to metaheuristics based. Besides all these, a varying range of artificial neural network based models along with their hybrids have also been developed. In this research study, a thorough review of model based techniques has been carried out for the purpose of getting details vis-à-vis strengths and weaknesses in these model based software cost estimation techniques.

Keywords: *Algorithmic Techniques, Model Based Techniques, Software Cost Estimation.*

I. INTRODUCTION

Software cost estimation is the summation of predictions of both building effort and calendar time used to develop a software project. The building effort includes the summation of working hours and the total of workers included in the process of soft project development. Just from the inception of software project development, organizations of this nature came across to the problem of poor estimations of development effort and development time of software projects. A good reason for this was and which is persistent even this time is the availability of vague information about the software project to be developed at the time of its estimation process. A better estimate of software product is the only thing that can let any software development project manager to evaluate the project progress, gives him / her good track of potential cost control and accuracy in delivery time. This in widespread, however, gives the organization a better insight of resource utilization and consequently will land the organization in a better schedule of its futuristic projects. For this purpose, a good number of software cost estimation techniques from last so many decades have been proposed which differ from algorithmic [1] [2] to Non algorithmic to Data mining and to metaheuristics algorithmic based, but unfortunately non among these satisfy the acceptance standard of accuracy in estimation of software development projects. However, a varying range of artificial neural network based models along with their hybrids have also been developed and have shown improvements in the said. The most commonly used method for predicting software development effort as the case with COCOMO [1] [3] [4] are based on linear-least-squares regression. Being extreme susceptible to local variations in data points [5], the model have failed in dealing with implied Non-linearities and interactions between the characteristics of the project and effort [6]. Moreover, software cost estimation models if yield estimates with 25% of mean relative error to the actual and if follows for at least 75% of the time, are believed to be in acceptable accuracy. However, there remains an open space for developing effort estimation models with better predictive accuracy [7] too.

II. MODEL BASED TECHNIQUES

In model-based techniques, inorder to determine estimation of cost interms of effort in person months and calendar time of any software project to be developed, a mathematical model is used containing some cost factors to decide this estimation. The functionality of the mathematical model is usually dictated by the concept and testing of the mathematical models made for estimation of the currently going software project. However, in most of the cases, these models also want their calibration with past data from historical projects.

The techniques of this kind are impartial and never get unfair by human factor. These techniques are iterative and help in adding sensitivity on the estimates of software to be developed. Moreover, once attuned with quality data from past similar projects, these models provide accurate results.

However, in contrast to above, model based techniques are not able to work in well once exposed to special conditions like when exposing them to poor quality of input data. Moreover, as these models are standardized using data from history projects, which cannot be certain for such models about how much these models prove the accuracy.

In the coming subsections, three famous model-based techniques are discussed in detail. These include Putnam model, also called as Software Life-cycle Model (SLIM), Constructive Cost Model, and Function Point Analysis. Some other model based techniques like Estimacs like [8] Checkpoint [9], SEER-SEM [10] and Price-S [11] are also found in the literature.

Putnam's Model

This model was given in 1970s by Larry Putnam [12]. In this model, the Putnam uses a Rayleign curve function to define estimates of both effort and time needed to complete a precise sized software development project. Software equation is the dominant share of Putnam's model and is given as:

$$S = E \times (Effort)^{1/3}\, xd^{4/3} \qquad (1)$$

, where '*xd*' represents the delivery time of software to be developed in years, '*E*' defines the environmental factor and reveals the capability of software development. '*S*' representing the size is the actual source lines of code (SLOC) and include altered as well as new coding lines. These coding lines are counted while excluding blank and comment lines. '*Effort*' represents the total effort in person-years applied to the software project. Another significant understanding given by Putnam includes the following:

$$Effort = y_0 \times td^3 \qquad (2\ )$$

, where 'y_0' represents a manpower build-up and varies from 8 to 27 software projects defining from entirely new to rebuilt softwares respectively. The softwares to be developed from the scratch, with known size and have to interact with other systems for which they need an interface will be having highest randomness and will take longer time to develop. Whereas in-case of reengineering of an existing software by doing its modification will be having very less entropy associated with them as a good share of the modules interms of code and logic have previously been developed. Thus such kind of softwares developments will need less time to develop too. On combination of the above two mathematical models represented by (1) and (2), a third model can be developed and is as represented as:

$$Effort = \left(y_0^{4/7} \times E^{-9/7}\right) \times S^{9/7}$$

and

$$x_d = \left(y_0^{-1/7} \times E^{-3/7}\right) \times S^{9/7}$$

The factor '*E*' commonly known as environmental factor can be substituted with another factor 'B', called as scaling factor.

Where scaling factor represents a function of project size and productivity of process represents the capability of a specific software company inorder to develop any software of any specific defect rate and known size. Moreover by using Putnam's Model, estimation of software development effort can be plotted as shown in below figure 1 to be the function of time. In this figure, various points on sideways of the curve signify total effort as estimated and completed at any particular time. From the below graph, it is clear that the curves depict that once a project enters into its extension with respect to the completion, the total effort decreases correspondingly. This feature of Putnam's model is considered to be its unique one.

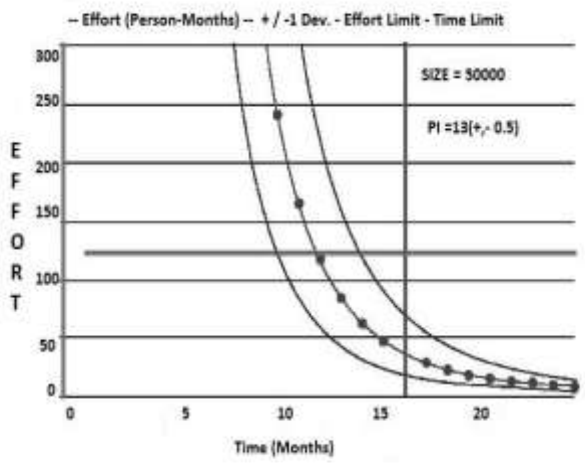

Figure 1: Time Function in Putnam Model with respect to Software Development Effort

The Putnam model can be adjusted with some adjustment questions if there exists non availability of data from completed projects of past. So in conclusion, simplicity interms of calibration of this model with history data becomes its significant property.

Function Point Analysis

This model was developed by Albrecht [13] at IBM as a way to measure the amount of functionality in a system. They are derived from the requirements. Unlike lines of code, which capture the size of an actual product, function points do not relate to something physical but, rather, to something logical that can be assessed quantitatively. As shown in Figure 3.2, the function-point metric is calculated in two steps. First, a table like Table 1, which captures both data and transaction information, is used to calculate an initial function point count.

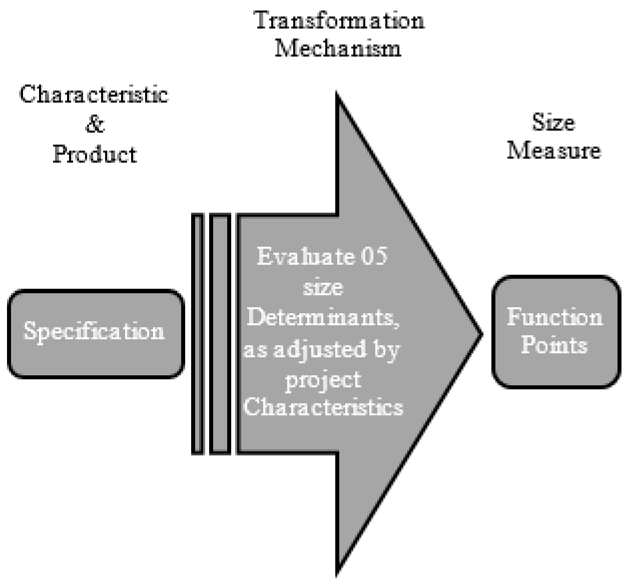

Figure 2: Transformation of Requirements into Function or Feature Points

Table 1: Initial Function-Point Count

Input Type	COUNT		
	Simple	Average	Complex
Number of External Inputs	3	4	6
Number of External Outputs	4	5	7
Number of External Queries	3	4	6
Number of Internal Logical Files	7	10	15
Number of External Interface Files	5	7	10

For each of the row's count in column second is simply multiplied by a suitable weight as given in column 3, column 4, and column 5 to produce a numerical value that represents the mandatory quantity of functionality backed by that row. These five numbers (weighted) are later summed to produce an "unadjusted function point" count.

In second step adjustment of initial count is done by using features that let the project to be either extra or less challenging than a typical one. The project is next categorised by making use of a six-stage scale with no influence represented with a weight of 0, incidental represented with a weight of 1, moderate represented with a weight of 2, average represented with a weight of 3, significant represented with a weight of 4 and essential represented with weight of 5 inorder to answer every question as given below in a set of 14 questions:

1. Whether the system needs reliable recovery and backup?
2. Does the system require data communications?
3. Are there processing functions distributed?
4. Is performance needed to be critical?
5. Is there any possibility of the system run in under heavily loaded operational environment?
6. Does the system need on-line data entry?
7. Does the data entry, if on-line, need the input transaction to be developed over many operations or screens?
8. Is there online Updation of master files?
9. Are the files, inquiries, inputs and outputs to be complex?
10. Whether the internal processing complex?
11. Whether code is reusable?
12. Does installation and conversion to be part of design?

13. Whether the system made for different organizations with several installations?
14. Whether the application is made to simplify variation and user-friendliness by the user?

The weights are next applied to unadjusted function-point total to produce an adjusted function-point count.
Albrecht and Gaffney in 1983 [14] interpreted function points to number of code lines for making their use in software cost estimation and software schedule estimation models.
Later in 1986, function points at Software Productivity Research were enhanced by Capers Jones [15] and named as feature points. Software Productivity Research feature points augment one more algorithmic parameter to the existing five function-point parameters.

Although function and feature points find their greater use in providing size estimate in a schedule-estimation method, however, they are don't find any importance in determining the project's status. In other words, we can say that it is not genuine to say that a certain fraction of the system is done if that fraction of function points has been completed with respect to its coding, as extra effort is required to integrate together all the functions once completed.

Constructive Cost Model
A research study was carried out by Barry Boehm in 1970 on 63 TRW Aerospace software projects which later in 1981 turned into one of the famous algorithmic model [1] called as Constructive Cost Model and COCOMO 81 for short. This model fits best to the software development projects developed under the base of waterfall model and developed using Common business oriented and assembly like languages. The model was later improved in 1990 and became famous as COCOMO-II [2]. COCOMO-II however proves good in meeting the changing developing environment requirements.

COCOMO 81 (Constructive Cost Model) though uses mathematical formulae is the most commonly used software cost estimation model for estimating the development effort in person-months for a software project at different stages of software development life cycle. Thus COCOMO 81 because of its simplicity is taken as a base model for software cost estimation and thus necessitates an overview in the following section.

The COCOMO 81 model is a regression based model, used to calculate the amount of effort and the time schedule for software projects. It is considered to be the most cited best known and the most credible model among all traditional algorithmic models. COCOMO 81 though considered to be the most stable software cost estimation model of the time but fails to match the development environment of the late 1990's. So, in 1997 COCOMO II was published which gave solution to most of the problems lying with COCOMO 81. Both of the models will be briefed in next subsections:

Basic Cocomo
It is the fundamental software cost estimation model made to estimate software development cost of projects of the nature of small to medium size. Being simple and easy to use, this model provides quick cost estimates but this simplicity lets its accuracy to be limited by some measure, thus this model is normally used for rough and initial estimates of cost of software projects.

In this model, the function of program size is the value of the estimation of software development effort. Normally the size of program is determined as thousand lines of code. The basic COCOMO model is given using the following equation:

$$Effort\ (Person\ Months) = a_b \times (KDSI)^{b_b}$$
$$Development\ Time\ (in\ Months) = c_b \times (Effort)^{d_b}$$
$$Productivity = KDSI/Effort$$
$$Average\ Staffing = Effort/Development\ Time$$

The coefficients 'a_b', 'b_b', 'c_b', 'd_b' are defined per project mode and they are given in the following table:

Table 2: Basic COCOMO Coefficients

Software Project	a_b	b_b	c_b	d_b
Organic	240	105	250	38
Semi-attached	300	112	250	35
Embedded	360	120	250	32

However, there exist so many factors of cost which are excluded by basic COCOMO. These cost factors include Constraints interms of hardware, Capability, experience and capacity of the personnel involved in overall software development process.

Intermediate Cocomo

The software development effort estimation using intermediate COCOMO is a function of two things namely 15 attributes of cost factors and program size. These 15 attributes are categorised into following four groups:

1. Attributes related to Software product,
2. Attributes related to Machine hardware,
3. Attributes related to People involved,
4. Attributes related to overall project.

Moreover associated with each attribute is a weight called as multiplying factor which determines the effect of attribute involves in estimation process of software development effort.

Estimation of software development cost in this kind of COCOMO initiates by developing a nominal estimate of effort and making use of similar basic COCOMO scaling equations. Next the nominal estimate is accustomed by putting multipliers of effort drawn from the ratings of the software project. The ratings of the software project are regarding 15 cost drivers and every single of these attributes get a score from the range of very low to extra high. The various multipliers of effort which are applied to ratings are given in the following table 3.3, whose product result in calculation of Effort Adjustment Factor (EAF). The common values of EAF are from 0.9 to 1.4.

Table 4: Effort Multipliers in Intermediate COCOMO

Attribute Categories	Cost Drivers	RATINGS					
		Very Low	Low	Nom.	High	Very High	Extra High
Product	Software Reliability Requirement	75	88	100	115	140	-
	Application Database Size	-	94	100	108	116	-
	Product Complexity	70	85	100	115	130	165
Hardware	Runtime Performance Constraints	-	-	100	111	130	166
	Memory Constraints	-	-	100	106	120	156
	Virtual Machine Environment Volatility	-	87	100	115	130	-
	Mandatory turn of time	-	87	100	107	115	-
Personal	Capability of Analyst	146	119	100	86	71	-
	Applications Experience	129	113	100	91	82	-
	Software Engineering Capability	142	117	100	86	70	-
	Experience of Virtual Machine	121	110	100	90	-	-
	Programming Language Experience	114	107	100	95	-	-
Project	Application of Software Engineering Methods	124	110	100	91	82	-
	Software tools usage	124	110	100	91	83	-
	Development Schedule Requirement	123	108	100	104	110	-

Intermediate COCOMO takes the following form:

$$Effort = a_i \times (KLOC)^{b_i} \times EAF$$

, where '*E*' represents the effort in person-months,
'*KLOC*' represent the predicted lines of code of the project and
'*EAF'* defines the effort adjustment factor to be computed centred on cost drivers.

However, the use of '*E*' in calculating '*D*' (Development time) is done in the similar manner as in Basic COCOMO Moreover, exponent 'b_i' and coefficient 'a_i' are shown below in table 3.4:

Table 3.4: Intermediate COCOMO Coefficients

Software Project	a_i	b_i	c_i	d_i
Organic	320	105	250	38
Semi-attached	300	112	250	35
Embedded	280	120	250	32

Detailed Cocomo

Detailed COCOMO incorporates all characteristics of the Intermediate COCOMO with an assessment of the cost driver's influence on individual project phases. This is done by using different efforts multipliers for each cost drivers attribute in each phase. These multipliers are called Phase Sensitivity Effort Multipliers and these determine the amount of effort required to complete each phase of the project.

Cocomo II

As earlier stated that COCMO 81 fails to match the development environment of the late 1990's provides space for COCOMO II to get published which gave solution to most of the problems lying with COCOMO 81.
COCOMO II has three models and they include:

1) Application Composition Model – This model is based on new Object Points and thus fits to projects that are developed using modern GUI-builder tools.
2) Early Design Model – It is an Unadjusted Function Points or KSLOC based model and is used to generate rough estimates of a project's cost and duration prior to the determination of projects entire architecture. It uses a small set of new cost drivers, and new estimating equations.
3) Post-Architecture Model – This model is used once the overall structural design for a software project is completed. This model is the detailed one among all the three uses function points or LOC as size estimates and engrosses the actual development and maintenance of a software product.

COCOMO II considers17 cost drivers, used in the Post Architecture model. These cost drivers are rated on a scale from Very Low to Extra High in the same way as they are followed in COCOMO 81. COCOMO II post architecture model is given as:

$$Effort = A \times [SIZE]^B \times \prod_{i=1}^{17} Effort\ Multiplier_i$$

$$where\ B = 1.01 + 0.01 \times \sum_{j=1}^{5} Scale\ Factor_j$$

In above equation, '*A*' represents Multiplicative Constant, '*SIZE*' represents Size of the software project measured in terms of the selection of scale factors (SF) and is based on the rationale that they represent a significant source of exponential variation on a project's effort or productivity variation.

Table 3.5: COCOMOII- Effort Multipliers

Attribute Category	Cost Drive
Product	Requirement of Software Reliability
	Size of Database
	Complexity of Product
	Reusability Required
	Document Match
Platform	Constraints of Execution Time
	Constraints of Main Storage
	Platform Volatility
Personal	Programmer & Analysts Capability
	Application Experience
	Platform Experience
	Language & Tools Experience
	Personal Continuity
Project	Modern Programming Practice Usage
	Software Tools Usage
	Multisite Development
	Development Schedule Requirement
	Security Application Classification

The following equation (i) represents the equation of effort for COCOMO II:
$Effort\ in\ person\ months = A \times Size^{E} \times \prod_{i=1}^{17} EM_i$,
where
'*Size*' represents function points or source line of code which are measured in Kilo source lines of code (KSLOC).
'*A*' is a constant and defines a factor of adjustment and depicts productivity dimension with a standard value of 2.9.

The factor '*E*' called as scale factor is based on following five factors:

- Flexibility in Development,
- Flexibility of Architecture,
- Resolution,
- Cohesion of a team,
- Process precedentedness and their maturity.

There is an exponential impact of SF on software development projects effort. The Scale Factors perform the same way as Effort Multipliers in contributions to Cost Factors.

III. CONCLUSION

The main concern of any software product client is to procure a facility that is able to meet its functional requirements, of the required quality, and delivered within an acceptable budget and timeframe. Cost estimates prepared in the early stages of a software development projects allow the clients to perform a cost-benefit analysis, secure funding as well as used as a basis for cost control during project delivery. Where the software product is a commercial asset, the initial capital investment must be balanced with the cost of maintenance and operations over the life-time of the software product to ensure that the project remains profitable and planned returns of capital investment are achieved over an estimated period. Decisions made at the early stages of the software development project therefore carry far-more reaching economic consequences and can seal the financial fate of any software development organization.

IV. FUTURE SCOPE

Researchers into the development of software cost prediction systems have now recognized the fact that there are many alternatives to algorithmic and other non-parametric of software cost estimation which may include data mining to machine learning models. However for the effective potentiality of any of these techniques to be realized it is important that they are brought closer to the estimation practitioner. For example, in the case of neural networks there is a certain level of competence required before they can be deployed effectively by a practitioner as it becomes complex with respect to their trainings and other architectural complexities to adopt them which sums to their lacking of desired competence . So to bring this competence or to replace these with their simple alternatives without the loss of any of their estimating power, any amount of research proving their accuracy will not encourage practitioners to embrace them. In this research, the competence into these artificial neural network based models is brought by their inclusion with various metaheuristic optimization algorithms like artificial bee colony, ant colony, fire fly, particle swarm optimizations and many alike that either on their alone or their hybridisation with them or with their higher end artificial neural network based models like functional link artificial neural networks offer a number of potential advantages over their counterparts.

REFERENCES

1. *Boehm, B.W., "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, New Jersy, 1981.*
2. *Boehm, B.W., "Software Cost Estimation with COCOMO II", Prentice Hall PTR, Englewood Cliffs, New Jersy, 2000.*
3. *Fenton, N.E., Pleeger, S.L., "Software Metrics: A Rigorous and Practical Approach", second ed. PWS, Boston, MA, USA, 1997.*
4. *Pressman, R.S., "Software Engineering: A Practitioner's Approach", fourth ed. McGraw-Hill, New York, NY, USA, 1997.*
5. *Gray, A.R.,. "A simulation-based comparison of empirical modelling techniques for software metric models of development effort," In: Proceedings of ICONIP, Sixth International Conference on Neural Information Processing, Perth, WA, Australia, vol. 2, pp. 526–531, 1999.*

6. *Kemerer, C., "An empirical validation of software cost estimation models". Communications of the ACM 30 (5), 416–429, 1987*
7. *Rubin, H., "Interactive macro-estimation of software life cycle parameters via personal computer: a technique for improving customer/developer communication", in Proc. Symp. on application & assessment of automated tools for software development, IEEE, San Francisco, 1983.*
8. *Jones, C., "Applied Software Measurement", McGraw Hill, 1997.*
9. *Jensen, R., "An Improved Macro level Software Development Resource Estimation Model", Proceedings 5th ISPA Conference, pp. 88-92, April 1983.*
10. *Freiman, F.R. and Park, R.E., "The Price software cost model": RCA government systems division IEEE, 1979.*
11. *Putnam, L., "A general empirical solution to the macro software sizing and estimating problem", IEEE Transaction on Software Engineering, pp. 345-361, July 1978*
12. *Albrecht, A. J., "Measuring application development productivity", Proceeding of the Joint SHARE, GUIDE and IBM application development symposium, IBM Corporation, 1979.*
13. *Albrecht , A.J., and Gaffney, J.E., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Trans. Software Eng., vol. 9, no. 6, pp. 639-648, Nov. 1983.*
14. *Capers, Jones," Software Productivity Research" (http://www.spr.com), 1986.*