Analysis Of the Techniques for Software Cost Estimation

Poonam Pandey

Assistant Professor ,GLA University ,Mathura

Email-poonam.pandey@gla.ac.in

***Abstract*: One of the most valuable asset in any software industry is the correct estimation of effort and hence cost estimation of the software to be developed by them. Because of the highly dynamic nature of the Software development , it becomes more and more difficult to get a correct software effort estimation and software cost estimation, which is one of the most important factor which makes software more competitive and is essential for controlling Software Development Cost. Software Cost Estimation is one of the challenging managerial activity, because values of many of the variables are not known and not easy to predict at an early stage of Software Development .An ideal Software Cost Estimation Model should provide ample confidence, precision and accuracy from its predictions. In this paper, we have performed an analysis of most of the algorithmic techniques which has been developed till now for Software Cost Estimation. We have also tried to analyze the advantages and shortcomings of every technique.**

Keywords: Software Cost Estimation, COCOMO model, parametric models, Non-parametric models.

I. INTRODUCTION

If we just have a look at the development of global software market, we can easily observe the declination in hardware costs and a tremendous increase in the computing capacity, but still the cost of developing software has increased significantly. Software Development is effort intensive procedure. For managing any software project effectively, it is very crucial to estimate the involved Software Cost and Effort accurately. Predicting an accurate Software Cost Estimation and Software Effort Estimation is one of the most challenging and demanding task in every software industry. For the projects which are of high budget, very big, important, and have temporal constraints, having an accurate Software Cost and Effort Estimation becomes more and more important and at the same time predicting an accurate Software Cost and Effort Estimation becomes more difficult also for such projects. If we categorize the software projects into three categories i.e. small projects, medium projects, large projects then we can say that chances of having an accurate estimate of software cost and effort for them will be high ,medium and low respectively. This is because, with the increase in the complexity of software, it becomes more difficult to have an accurate estimate of software cost and effort. Following graph depicts this relationship between complexity of software projects and chances of having an accurate estimate of software cost and effort Cost estimates of software projects can be used by management for evaluating those projects' proposal and for effective management of the Software Development Process. Hence, we can say that the correct approximation of cost of software development has a large economic influence: in fact, approximately 60% of large projects exceed their Software Cost Estimates and the projects which are never completed because of the false estimation of Software development Cost [l].Basically effort is the main driving factor for the Cost, and because of this fact, estimation of Software Cost is ultimately a problem of Software Effort estimation. Number of scientists and researchers are trying their best for developing more and more accurate, new software cost estimation techniques [5, 6, 7].Most of the software cost estimation techniques are based on the algorithmic models, expert judgment, or machine learning methods.

II. FACTORS AFFECTING SOFTWARE COST DEVELOPMENT

There are basically, two factors which affect the project costs:

- The efficiency of the implementation
- The cost of the studies done prior to software development.

The above two factors are very dynamic and vary from project to project. There are many factors which are to be considered while software cost estimation.

III. TECHNIQUES PROPOSED FOR GETTING ACCURATE SOFTWARE COST ESTIMATE:

A myriad of techniques have been developed till date for making

978-0-7695-4941-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ACCT.2013.13

accurate Software Cost estimates and for avoiding misestimations Broadly ,we can classify all these techniques into the following two major categories:

(1)Parametric models.
(2)Non-parametric models.

Parametric models are derived from the statistical or numerical analysis of historical projects data [2, 3], Non-Parametric models include those techniques which are based on a set of artificial intelligence techniques such as artificial neural networks, analogy based reasoning, regression trees, genetic algorithms and rule-based induction[4,9,12,13,14,15].Most of the algorithmic model based techniques comes under parametric models and techniques based on expert methods, machine learning methods comes under non-parametric models.

Algorithmic Models: Numerous algorithmic models are available today to estimate software costs. Some of them are listed below:
(1)Boehm's COCOMO (Constructive Cost Model) model: It is the basic method which is used in the industry, in all over the world.

This model is used for predicting the required number of persons in every month for the development of software. The model is also able to give an estimate of the development schedule in months and with the help of this model we can also estimate that how much effort should be given to each and every major phases of software development.

Boehm proposed three levels of the model:

1) Basic COCOMO

2) Intermediate COCOMO

3) Detailed COCOMO

Majority of software projects apply Basic COCOMO model to estimate the cost of Software Development. What Boehm says about the model is : "Basic COCOMO is good for rough order of magnitude estimates of software costs, but its accuracy is necessarily limited because of its lack of factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes known to have a significant influence on costs."

The Software Cost Estimation is done with the help of this model by using one of the three development modes:

1) Organic,

2) Semidetached

3) Embedded.

Boehm describes the modes as follows:

Organic: In the organic mode, relatively small software teams develop software in a highly familiar, in-house environment. Most people connected with the project have extensive experience in working with related systems within the organization, and have a thorough understanding of how the system under development will contribute to the organizations objectives. Very few organic-mode projects have developed products with more than 50 thousand delivered source instructions (KDSI).

Semidetached: The semidetached mode of software development represents an intermediate stage between the organic and embedded modes. The size range of a semidetached mode product generally extends up to 300 KDSI.

Embedded: The major distinguishing factor of an embedded-mode software project is a need to operate within tight constraints. The product must operate within (is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures, such as an electronic funds transfer system or an air traffic control system.
Examples of other algorithmic models include the following :
(2)SDC model (Nelson, 1966)
(3) TRW Wolverton model (Wolverton, 1974)
(4) Putnam **SLIM (Software Lifecycle Management)** model (Putnam, 1978)
(5)Doty model (Herd ***et al.,***1977)
(6)RCA PRICE **S** model (Freiman and Park, 1979)
(7)IBM-FSD model (Walston and Felix, 1977)
(8)Boeing model (Black *et* ***al.,*** 1977)
(9)GRC model (carriere and Thinodeau, 1979)
(10)Meta model (bailey and Basili, 1981).

Using Algorithmic models provides the advantage that we work with an economically feasible approach for estimating software costs, but if we have a glance at the darker side of using Algorithmic models , then we conclude the following shortcomings of these models:

First one is that , the cost and effort estimates derived from different algorithmic models generally vary significantly from each other(Saiedian, Band, and Barney,1992).Variations of such kind causes problems to managers because it becomes difficult for them to decide the nearly exact amount of resources to be needed. Second point is that, the models are based on historical data and hence are not able to reflect current progresses in the areas of programming languages, hardwares, and software engineering.

Expert Judgment based Methods : Generally ,historical data is not needed for expert judgment. Expert judgment is often based on management or estimator recollection of past projects that may or may not have been documented. Heemstra and Vigder concludes that 62% of estimators in organizations use this technique. Hughes has discussed and identified in his paper the strengths and weaknesses of expert judgment and acknowledged the usefulness of the method in industry [9]. Among the benefits of using expert judgment is that the estimation is customized to the specific organizational culture, hence this estimation technique is more accurate in comparison to general Algorithmic approaches. Expert judgment is a non-structured process even though in many cases it has been proven to give a better accuracy than using other techniques. The final estimation of the experts is subjective and based on feelings and logic. The logical aspect of human decision-making process is very complex but it is based on the process of copying and mimicking [8].

Machine Learning Methods:

Neural Network Model

There are a number of learning algorithms to train neural networks, among which the back propagation (back-prop) paradigm has been considered to be the most popular learning mechanism for all problems based on prediction and classification. Every back-prop neural network **is** always organized in layers. Each layer is composed of neurons which is also called processing elements and their connections (Rummelhart et al. 1986).
Basically there are three layers:

- Input Layer: It is the first layer which contains neurons that represent the set of input variables.
- Output Layer: The output layer contains neurons that represent the output variables.
- Hidden Layer: Nonlinear relationship between the input layer and output layer variables are dealt with the help of hidden layer. The hidden layer helps in the extraction of higher level features and for facilitating generalization.

Numerical weights are associated with the connections between neurons. Examples from the training set are repeatedly fed in the training process for adjustment of these numerical weights which are associated with the connections between neurons. An activation level is also associated with each neuron, which is specified by continuous or discrete values. Internal activation value which comes out into a neuron in the hidden or output layers is the sum of each incoming activation level multiplied by its respective connection weight. In the input layer, the activation levels of the neurons are determined by the response to the input signals received from the environment. Now ,the internal activation value is updated by a transfer function and this now becomes an output, which in turn may become an input to one or more neurons. A sigmoid transfer function is typically used to transform the input signals into output signals specifically for classification problem. The sigmoid transfer function is represented by F(I) = **1/(** l+e^-1), where *I* represents the internal activation. For any output error, all the connection weights are assumed to be responsible in a back propagation network. The difference between a network's estimated output or predicted value and the corresponding observed output value is known as error. The error values are calculated at the output layer and propagated to previous layers and used for adjusting the connection weights. The training process consists of repeatedly feeding input and output data from empirical observations, propagating the error values, and adjusting the connection weights until the error values fall below a user-specified tolerance level. Equations are there which describe how the error values are computed and the connection weights are updated:

Fuzzy Logic and Evolutionary Algorithms, Genetic Programming : These are believed to provide nearly accurate approximations. When we talk in terms of accuracy, then Genetic Programming was found more accurate than other techniques, but its drawback is that it does not converge to a good solution as consistently as Neural Network. Thus we can say that more emphasis is to be provided on defining that what are the measures, and combination

of measures, which will be more suitable for the specific problem. In Genetic Programming, generally different datasets are used and hence it yields diverse results, which are classified as acceptable, moderately good, moderate and bad results.
The datasets examined vary extremely in terms of size, complexity, homogeneity, therefore it is not easy to obtain granularity consistent results.

IV. CONCLUSION

Software cost estimation is a complex activity because of the numerous cost drivers, which display more than a few value discrepancies between them, and also because software cost estimation is highly affected by many factors. Both metrics, qualitative as well as the quantitative measures are included in the Software development metrics for a project. Qualitative metrics include experiences and skills of team members, environment in which software development takes place, culture and group dynamism of organization and quantitative measures include size of project, resources. However all these measures vary greatly for every project and generally these values are ambiguous, dissimilar and vague for every project. There are no formal guidelines for determining the actual effort which is required for completing a project based on specific characteristics and attributes. Previous attempts to identify possible methods to accurately estimate development effort were not as successful as desired, mainly because calculations were based on certain project attributes of publicly available datasets [11]. Nevertheless, the proportion of evaluation methods employing historical data is around 55% from the total 304 research papers investigated by [10] in 2004. Overall we conclude that the most important thing is that datasets of the current and future projects which are required during the evaluation of estimation methods should be as representative as possible. All methods proposed have their own advantages and disadvantages. None of the factors which affect the project cost and development should be ignored while estimating the Software development Cost.

REFERENCES

[l] **M.** Boraso, C. Moutangero, and B. Sedehi. Software cost estimation: an experimental study of model performances. Technical Report **TR-96-22,** DEPARTIMENTO DI INFORMATATICA, UNIVERSITA DI PISA, Italy, 1996.

[2] B.W. Boehm. Software Engineering Economics. Prentice-Hall, 1981

[3] B.W. Boehm et al. "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0." Annals of Software Engineering on Software Process and Product Measurement, Amsterdam, 1995.

[4] C. J. Burgess and M. Lefley. "Can Genetic Programming Improve Software Effort Estimation?" Information and Software Technology, vol. 43, 2001, pp. 863-873.

[5] J.P. Lewis, “Large Limits to Software Estimation,” *Software Engineering Notes*, Vol. 26, No. 4, July 2001.
[6] Stamelos, etal. "Estimating the development cost of custom software", Information and Management, v.40 n.8, pp. 729-741, 2003.
[7] R. W. Jensen, "Extreme Software Cost Estimating", CrossTalk, Journal of defense software Eng., Jan 2004.

[8] J. E. Zull, "The art of changing the brain", Stylus Publishing, LLC, 2002.

[9] R. T. Hughes, "Expert judgment as an estimating method", Information & Soft. *Technology*, pp. 67-75, 1996.

[10] M. Jorgensen, M. Shepperd, “A Systematic Review of Software Development Cost Estimation Studies”, *IEEE Transactions on Software Engineering*, Vol. 33, No. 1, IEEE
Computer Press, Washington D.C., 2007, pp. 33-53.
[11] E.S. Jun, J.K. Lee, “Quasi-optimal Case-selective Neural
Network Model for Software Effort Estimation”, *Expert Systems
with Applications*, Vol. 21, No. 1, Elsevier, New York, 2001, pp.
1-14.

[12] M. Shepperd and C. Schofield. "Estimating Software Project Effort Using Analogies." Transactions on Software Engineering, vol. 23, no. 12, 1997, pp. 736-747.
[13] K. Srinivasan, and D. Fisher "Machine Learning Approaches to Estimating Software Development Effort." IEEE Transactions on Software Engineering, vol. 21, no. 2. February, 1995, pp. 126-136.

[14] S. Vicinanza, and M.J. Prietolla, "Case-Based Reasoning in Software Effort Estimation." Proceedings of the 11th nt. Conf. on Information Systems, 1990.
[15] G. Wittig and G. Finnie. "Estimating Software Development Effort with Connectionist Models." Information and Software Technology, vol. 39, 1997, pp. 469-476.