Asian Journal of Multidisciplinary Studies
Volume1, Issue.1, August 2013 Online available at www.ajms.co.in
ISSN: 2321-8819

An Overview of Software Cost Estimation Models

Suvarna R.Jagtap
Assistant Professor
BharatiVidyapeeth Deemed University
YashwantraoMohiteCollege,Pune- 38
Email:- suvarna.rjagtap@gmail.com

Abstract:-

To date most work carried out in the software cost estimation field has focused on algorithmic cost modeling. In this process, costs are analysed using mathematical formulae linking costs or inputs with metrics to produce an estimated output. The formulae used in a formal model arise from the analysis of historical data. The accuracy of the model can be improved by calibrating the model to your specific development environment, which basically involves adjusting the weightings of the metrics. There are a variety of different models available, the best known are Boehm's COCOMO[BOEHM-81], Putman's SLIM , and Albrecht's' FP [ALBR-83].This paper takes an overview of various Software Cost Estimation Models used widely for software project cost estimation.

Keywords: algorithmic cost model ,metrics,COCOMO,SLIM ,FP

Introduction:-

Software project development includes a number of activities that result in a delivered product (software). As software becomes more and more expensive to develop, project management has been recognised as a difficult task in practice. There are a lot of unpredictable factors existing in the software development cycle that have become contributing factors to this problem.Project planning is basically a 12 step program which includes :-

1) Set goal and scope
2) Select lifecycle
3) Set org./team form
4) Start team selection
5) Determine risks
6) Create WBS
7) Identify tasks
8) Estimate size
9) Estimate effort
10) Identify task dependencies
11) Assign resources
12) Schedule work

On an initial instinct you might expect formal models to be advantageous for their 'off-the-shelf' qualities, but after close observation this is regarded as a disadvantage by cost estimators due to the additional overhead of calibrating the system to the local circumstances. However, the more time spent calibrating a formal model the more accurate the cost estimate

should be. A distinct disadvantage of formal models is the inconsistency of estimates,[KEMERER] conducted a study indicating that estimates varied from as much as 85 - 610 % between predicated and actual values. Calibration of the model can improve these figures, However, formal models still produce errors of 50-100%. In terms of the estimation process , nearly all algorithmic models deviate from the classical view of the cost estimation process.In terms of the estimation process , nearly all algorithmic models deviate from the classical view of the cost estimation process.

Figure 1.0 : Classical view of the algorithmic cost estimation process

An input requirement of an algorithmic model is to provide a metric to measure the size of the finished system. Typically lines of source code are used, this is obviously not known at the start of the project. SLOC is also very dependent on the programming language and programming environment, this is difficult to determine at an early stage in the problem especially as requirements are likely to be sketchy. Despite this SLOC has been the most widely used size metric in the past, but current trends indicate that it is fast becoming less stable. This is probably due to the changes in software development process in recent years highlighted with a tendency to use prototyping, case tools and so forth. An alternative is to use function points proposed by [ALBRECHT], which are related to the functionality of the software rather than its size. A more recent approach is to use object points. This is in comparison a new methodology and has not been publicised in the same depth as function points and SLOC. In essence the method is very similar to function points but counts objects instead of functions. Its recent rise has been prompted by the interest in the object orientation revolution.

Algorithmic models generally provide direct estimates of effort or duration. As shown in figure 1 the main input is usually a

prediction of software size. Effort prediction models take the general form :

effort = p*S

(1/productivity rate)

where**p** is a productivity constant and **S** is the size of the system.

E.g. productivity = **450** source lines of code per month, making **p = 0.0022** and the size of the system has been estimated at **8500** KLOC.

effort=0.0022 * 8500;**effort = 18.7 person moths**

The example above assumes that the relationship between effort and size is a linear one. Most models allow for non-linear relationships by introducing economies or dis-economies of scale. The general formula being:

effort = p * Se

These findings indicate that there is greater productivity when building large software systems as opposed to small systems. However, the results can be justified as it is expected that larger teams can specialise and the overheads are of a relatively fixed size.

There are various estimation methodologies used for software project estimation. They are as follows:-

Table 1.0 :Cost Estimation Methodologies

TOP DOWN	BOTTOM UP	EXPERT JUDGEMENT	ESTIMATION BY ANALOGY	PRICED TO WIN
Based on overall characteristics of project-Some of the others can be "types" of top-down (Analogy, Expert Judgment, and Algorithmic methods) **Advantages** Easy to calculate Effective early on (like initial cost estimates) **Disadvantages** Some models are questionable or may not fit Less accurate because it doesn't look at details	Create WBS Adds from the bottom-up **Advantages** Works well if activities well understood **Disadvantages** Specific activities not always known More time consuming	Use somebody who has recent experience on a similar project get a "guesstimate" Accuracy depends on their 'real' expertise Comparable application(s) must be accurately chosen Systematic Can use a weighted-average of opinions	Use past project Must be sufficiently similar (technology, type, organization)Find comparable attributes (ex: # of inputs/outputs) Can create a function **Advantages** Based on actual historical data **Disadvantages** Difficulty 'matching' project types Prior data may have been mis-measured How to measure differences – no two exactly same	Just follow other estimates Save on doing full estimate Needs information on other estimates (or prices) Purchaser must closely watch trade-offs Priced to lose?

COCOMO(Algorithmic Method)

The best known and most transparent cost model COCOMO (Constructive Cost Model) was developed by [BOEHM], derived from the analysis of 63 software projects.It has evolved into a more

comprehensive estimation model called COCOMO II. Boehm proposed three levels of the model; basic, intermediate, detailed.As with all estimation models, it requires sizing information and accepts it in three forms: object points, function points, and lines of source code.

SLIM

Putman's SLIM (**S**oftware **LI**fe Cycle **M**anagement) is an automated 'macro estimation model' for software estimation based on the Norden/Rayleigh function. SLIM uses linear programming, statistical simulation, program evaluation and review techniques to derive a software cost estimate. SLIM enables a software cost estimator to perform the following functions:

1) Calibration : Fine tuning the model to represent the local software development environment by interpreting a historical database of past projects.
2) Build : an information model of the software system, collecting software characteristics, personal attributes, computer attributes etc.
3) Software sizing : SLIM uses an automated version of the lines of code (LOC) costing technique.

FUNCTION POINTS:

As an alternative to the problems identified with SLOC, [ALBRECT] devised a method of estimating effort by measuring the functionality of a system as opposed to size, namely function points. The approach taken is to identify and count a number of unique function types:

- external inputs (e.g. file names)
- external outputs (e.g. reports, messages)
- queries (interactive inputs needing a response)
- external files or interfaces (files shared with other software systems)
- internal files (invisible outside the system)

By focusing on the requirements specification document, the estimator can calculate the functionality of the system to be developed by identifying the function types listed above.

The sum of all the occurrences is computed by multiplying each raw function count with a weighting and then adding up all the values. The weights are based on the complexity of the feature being counted.

As an alternative to the problems identified with SLOC, [ALBRECT] devised a method of estimating effort by measuring the functionality of a system as opposed to size, namely function points. The approach taken

is to identify and count a number of unique function types:

- external inputs (e.g. file names)
- external outputs (e.g. reports, messages)
- queries (interactive inputs needing a response)
- external files or interfaces (files shared with other software systems)
- internal files (invisible outside the system)

By focusing on the requirements specification document, the estimator can calculate the functionality of the system to be developed by identifying the function types listed above.

The sum of all the occurrences is computed by multiplying each raw function count with a weighting and then adding up all the values. The weights are based on the complexity of the feature being counted.

CONCLUSION

Software Cost Estimation requires more effort from researchers to work on it as there is no technique which can show the precise or accurate result for cost estimation.So in order to get accurate cost estimation researcher, manager and organizations should work on it. Organization should work on completing every project data in future to get more accurate results.

REFERENCES :

1. Albrecht, A.J. and J.R. Gaffney, 'Software function, source lines of code, and development effort prediction: a software science validation', IEEE Trans. on Softw.Eng., 9(6),pp639-648, 1983.
2. Banker, R.D. and C.F. Kemerer, 'Scale economies in new software development', *IEEE Trans. on Softw.Eng.*, 15(10), 199-204, 1989
3. Boehm, B.W., Software Engineering Economics. Prentice-Hall: Englewood Cliffs, NJ, 1981.
4. Cowderoy, A.J.C. and J.O Jenkins, 'Cost estimation by analogy as a good management practise', in *Proc. Software Engineering 88*, ed. Pyle, I.C., Liverpool: IEE/BCS, pp80-84, 1988
5. DeMarco, T., Controlling Software Projects. Management, measurement and estimation. Yourdon Press: NY, 1982.
6. Fenton, N.E., 'Software Metrics: a rigorous approach'. Chapman & Hall,1991.
7. Fenton, N.E. and S. Pfleeger, 'Software Metrics: a rigorous and practical approach'. Thomson Computer Press, 1997.
8. Heemstra, F.J., 'Software cost estimation', *Information &Softw. Technol.*,34(10), pp627-639, 1992.
9. Hughes, R.T., 'Expert judgement as an estimating method', *Information &Softw. Technol.*, 38(2), pp67-75, 1996.
10. Jack R. and M. Mannion, 'Improving the software cost estimation process', *Software Quality Management.*, 1995 1 pp245-56.
11. Karunanithi, N., D. Whiyley and Y.K. Malaiya, 'Using neural networks in reliability prediction', *IEEE Softw.*, 9(4), 53-9, 1992.
12. Kemerer, C.F., 'An empirical validation of software cost estimation models', *CACM*, 36(2), 1993.

13. Kitchenham, B.A., 'Empirical studies of assumptions that underlie software cost estimation'. *Information and Softw. Technol.*, 34(4), 211-18, 1992.
14. Londeix, B., Cost Estimation for Software Development .Addison-Wesley:Workingham, 1987.
15. Londeix, B., 'Aspects of estimation practice in software development', in *Proc. Software Engineering 88*, ed. Pyle, I.C., Liverpool: IEE/BCS, pp 75-79, 1988
16. Low, G.C and D.R. Jeffery, 'Function points in the estimation and evaluation of the software process', *IEEE Trans. on Softw. Eng.*, 16(1), 64-71, 1990.
17. Low, G.C. and D.R. Jeffery, 'Calibrating estimation tools for software development', Softw. Eng. J., 5(4), pp215-221, 1990.
18. McDermid, J.A., Software Engineer's Reference Book, Butterworth-Heinemann: Oxford, UK, 1991.
19. Pengelly, A., 'Performance of effort estimating techniques in current development environments', Softw. Eng. J.l, September 1995, pp162-169
20. Putman, L.H., 'A general empirical solution to the macro software sizing and estimating problem'. *IEEE Trans. on Softw. Eng*., 4(4), 345-61, 1978.