International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

A Review paper on Software Effort Estimation Methods

Sangeetha K [1], Prof. Pankaj Dalal[2]

[1] M Tech Scholar (Software Engineering), Department of Computer Engineering, Shrinathji Institute of Technology & Engineering, Nathdwara-313301, India

[2] Associate Professor, Department of Computer Engineering, Shrinathji Institute of Technology & Engineering, Nathdwara-313301, India

Abstract

Software effort estimation is an important factor in any software industry. As software grew in size and complexity, it is very difficult to accurately predict the cost of software development. This was the dilemma in past years. The greatest pitfall of software industry was the fast changing nature of software development which has made it difficult to develop parametric models that yield high accuracy for software development in all domains. This paper summarizes several classes of software cost estimation models and techniques.No single technique is best for all situations, and that a careful comparison of the results of several approaches is most likely to produce realistic estimates. The use of workforce is measure as effort and defined as total time taken by development team members to perform a given task. It is usually expressed in units such as man-day, man-month, and man-year. This value is important as it serves as basis for estimating other values relevant for software projects, like cost or total time required to produce a software product. This paper reviews a research study comparing the different estimation techniques and illustrates the models using LOC and function point as an estimate of system size.

1. Introduction

An important aspect of any software development project is to know how much it will cost. The major cost factor is labour in most cases. Hence estimating development effort is centre to the management and control of a software project. Traditionally, effort estimation has been used for planning and tracking project resources. Effort estimation methods founded on those goals typically focus on providing exact estimates and usually do not support objectives that have recently become important within the software industry, such as systematic and reliable analysis of causal effort dependencies

It has been surveyed that nearly one-third projects overrun their budget and late delivered and two-thirds overrun their original estimates. The accurate prediction of software development cost is a critical issue to make the good management decisions. Also accurately determining how much effort and time a project required for project managers as well as system analysts and developers is important. Reasonably accurate cost estimation capability is needed by project managers to determine time and manpower for the project. The system analysts cannot make realistic hardware-software trade-off analyse during the system design phase due to unrealistic proposed budget and schedule. This may lead to optimistic over promising, hence the inevitable overruns, performance compromises as a consequence. But, actually huge overruns resulting from inaccurate estimates are believed to occur frequently.

Effort estimation is a critical activity for planning and monitoring software projects and for delivering the product on time within budget.

2. Research Study on Effort Estimation

Estimating the size, development time, and cost of software projects has largely been an intuitive process in which most estimators attempt to guess the number of modules, and the number of statements per module to arrive at a total statement count in the past era. Then, using some empirically determined cost per statement relationships, they arrive at a total cost for the software development project. Thus the traditional approach is essentially static. While this approach has been relatively effective for small projects of say less than 6 months duration and less than 10 man-years (MY) of effort, it starts to break down with larger projects and becomes totally ineffective for large projects

Many different studies have been published during the last 30 years comparing modelling techniques for software cost estimation.

Prior to 1970, estimation of effort was done manually by using Thumb rules or some algorithms which were based on Trial and error [2]. 1970 was an important period to predict the costs and schedules for software development. Automated Software cost estimating tools were built. Some difficulties were experienced in building large software systems [14]. During early 1970's the first automated software estimation tool had been built. This prototyping composite model is COCOMO (Constructive Cost Model) developed by Barry Boehm [2].

Function Point Analysis for estimating the size and development effort had been developed in 1975. 1977, PRICE-S Software estimation model was designed by Frank Freiman. It was the commercial tool to be marketed in United States. 1979, SLIM (Software Life Cycle Model) was introduced to US-Market by Lawrence H. Putnam based on Norden Rayleigh Curve [3]. Then The U.S. Department of Defence (DoD) introduced Ada programming language in 1983 and it reduced the cost of developing large systems. That model was named as Ada-COCOMO [15].1983, Charles Symons, a British software estimating researcher, introduced Mark II function point metric [16].1984, IBM had done a major revision of his function point metric which is basis of today's function points [2]. 1985, Caper Jones extended the concept of Function Point to include the effect of computationally complex algorithms [17]. 1986, IFPUG (International Function Point Users Group) was founded in Toronto, Canada due to rapidly growing usage of Function Point Metrics. 1990, Barry Boehm, at university of Southern California began to revise and extend the concept of original COCOMO model.

The researchers were comparedtime parametric models [1] [2] [3] [5] using data sets of various sizes and environments in these time. The main conclusions were that these models perform poorly when applied uncalibrated to other environments [4]. Kemerer et. al, used 15 projects from business applications and compared four models: SLIM [3], COCOMO [2], Estimacs [3], and Function Points (FP) [5]. He reported an estimation error in terms of the Mean Magnitude of Relative Error (MMRE) ranging from 85% to 772%.

Conte et al. used six dam sets from widely differing environments and reported an MMRE variation between 70% and 90% for their three tested models: SLIM [3], COCOMO [2], and Jensen's model. As a result of their investigation, they proposed a new model COPMO calibrated separately to the six data sets. This new model yielded a MMRE of 21%.

1993, the new version of COCOMO was introduced called COCOMO 2.0 which emerged in 1994 [18].1994, Rajiv D Banker and Hsihui Chang and Chris F Kemmerer, found it useful for cost estimation and productivity evaluation purposes' to think of software development as an economic production process [19].1996, Sophie Cockroft, obtained an accurate size estimations from the early system specifications [20]. 1997, Existing models were reviewed and more focus was on accuracy.1998, Chatzoglou constructed a new model called MARCS to give predictions of the resources (time, effort, cost, and people) [21].1999, J. J. Dolado, made a research about the estimation using the technique of Genetic Programming (GP) for exploring possible cost functions [22].

Then researchers also included non-parametric modeling techniques based on machine learning algorithms and analogy in the comparative study. Shepperd et al. [6] compared an analogy-based technique with stepwise regression. They used nine different data sets from different domains and report that, in all cases analogy outperforms stepwise regression models in terms of the MMRE.

Mukhopadyay et al. [7] used Kemmerer's project data set and found that their analogy-based model Estor, using case-based reasoning (CBR), outperformed the COCOMO model. Finnie et al. [8] compared CBR with different regression models using FP and artificial neural networks on a large database consisting of 299 projects from 17 different organizations. They report a better performance of CBR when compared with different regression models based on function points. In addition, artificial neural networks outperformed the CBR approach.

Srinivasan et al. [8] included regression trees, artificial neural networks, function points, the COCOMO model, and the SLIM model in their comparison. They used the COCOMO data set (63 projects from different applications) as a training set and tested the results on the Kemerer data (15 projects, mainly business applications). The regression trees outperformed the COCOMO and the SLIM model. They also found that artificial neural networks and function point based prediction models outperformed regression trees. Using a combination of the' COCOMO and the Kemerer data sets, Briand

et al. [9] compared the COCOMO model, stepwise regression, and Optimized Set Reduction (OSR), which is a non-parametric technique based on machine learning. OSR outperformed stepwise regression and the COCOMO model.

Jorgensen [10] used 100 maintenance projects for testing several variations of regression, artificial neural networks, and combinations of OSR with regression. He found that two multiple regression models and a hybrid model combining OSR with regression worked best in terms of accuracy. In general, he recommended the use of more sophisticated prediction models like OSR together with expert estimates to justify the investments in those models.

Although parametric techniques are included in almost all of the studies comparing different cost estimation methods, the comparisons are partial in the sense that only certain techniques are evaluated. Moreover, replications of studies are rarely performed. Even when the same data set is used in different studies, the results are not always comparable because of different experimental designs. Briand et al. and Srinivasan et al., for example, both used the COCOMO and Kemerer data; however, they used the data in different ways as training and test sets [9][8]. Furthermore, many studies used only small data sets coming from different environments. This makes it difficult to draw generalizable conclusions about the models' performance.

Kitchenham and Kansala [11] also note that better results can be obtained through disaggregating the components of function points and using stepwise regression to re estimate weights and determine the significant components.

2001, a new approach was proposed based on reasoning by analogy and in that linguistic quantifiers were used to estimate the effort [23].2002, Jorgensen, expert estimation was the most frequently applied estimation strategy for software projects [24]. 2003, Yunsik Ahn, Jungseok Suh, Seungryeol Kim and Hyunsoo Kim, they discussed software maintenance and proposed SMPEEM (Software Maintenance Project Effort Estimation) [25].

2007, different methods were introduced for estimating the effort. The average accuracy of expert judgment based effort estimates was higher than the average accuracy of models [26].2008, Parvinder S. Sandhu, et.al. focused on predicting the accuracy of models, as Neuro-Fuzzy system was able to approximate the non-linear function with more precision. So, neuro-fuzzy system was used as a soft computing approach to generate the model [27]. During 2009, some theoretical problems were identified that compared estimation models. It was invalid to select one or two datasets to prove validity of a new technique [28].2010, different estimation techniques were combined to reduce the error and keep control over the deviation of estimates away from actual [29, 30].

2011, many estimation techniques were proposed and used extensively by practitioners for use in Function Oriented Software development [31].2012, there were many software size and effort measurement methods proposed in literature, they were not widely adopted in practice. A lot of commercial software costs estimating tools have been released till today [32].

Although, most research into project effort estimation has adopted an algorithmic approach, there has been limited exploration of machine learning or non-algorithmic methods. For example, Karunanithi et al. [12] reported the use of neural nets for predicting software reliability, and conclude that both feed forward and Jordan networks with a cascade correlation learning algorithm outperform traditional statistical models. More recently Wittig and Finnie [13] describe their use of back propagation learning algorithms on a multilayer perceptron in order to predict development effort. An overall error rate (MMRE) of 29% was obtained which compares favourably with other methods. However, it must be stressed that the datasets were large (81 and 136 projects, respectively) and that only a very small number of projects were withdrawn for validation purposes.

3. Effort Estimation

Estimating is the process of forecasting or approximating the time and cost of completing project deliverables or the task of balancing the expectations of stakeholders and the need for control while the project is implemented.

The general form of effort estimation in any model can be:

$E = aS^b$

where 'E' is effort, S is size typically measured as lines of code (LOC) or function points, 'a' is a productivity parameter and 'b' is an economies or diseconomies of scale parameter.

A fundamental problem of software effort estimation is the determination of software size. The different approaches to measure software size are: Line of Code (LOC), function point (FP), use case point (UCP).

3.1 LOC-based models

A widely known model based on LOC is the constructive cost model (COCOMO) (Boehm, 1981). COCOMO classifies projects into three broad categories: organic or simple, semidetached or average, and embedded or complex. The COCOMO model itself has three versions. The intermediate version of COCOMO first calculates a nominal effort estimate in worker-months (WM) using a non-linear function based on the size of the software measured in thousands of delivered source instructions (KDSI):

$$WM = \alpha (KDSI)^{\beta}$$

where the values of the constants α and β are different for organic, semidetached, and embedded projects. Next, it adjusts the nominal estimate by multiplying WM by the ratings on 15 "cost drivers" that include attributes of the product, computer, personnel, and project. The COCOMO basic model, however, does not use any cost drivers. The COCOMO detailed model, divides the project into four phases (product design, detailed design, coding/unit test, and integration test) and estimates and applies the 15 cost drivers to each phase separately rather than to the entire project. Other models based on non-linear functions of LOC include the Doty model (Herd, et al., 1977) and the meta-model by Bailey and Basili (1981).

Another set of LOC-based models uses standard distributions as the basis for modeling the phase distribution of effort. Putnam's SLIM model, for example, determines the life cycle effort (K) in worker years based on number of source statements (S) (Putnam, 1978):

$$K = S^{3} C^{-3} t_d^{-4}$$

where t_d represents the time of peak manpower deployment and C is a technology constant. SLIM uses the Rayleigh curve to model the distribution of effort over time. The Jensen model also uses the Rayleigh curve for effort distribution (Jensen, 1983).

A criticism of the LOC-based models is that they require estimating LOC before developmentbegins. However, accurate LOC estimates may not be available until after the detail design is complete. The focus on LOC as an indicator of size also leads to problems when a model calibrated for one coding language is used for another without recalibration. Finally, variations in line counting methods may change LOC by a wide margin (Jones, 1986b).

3.2 Function point-based models

The function point method first assigns a weight to each unique input type, output type, logical file, external interface file, and external query handled by an application to reflect the "level of complexity." The total score for all function types, called the function count, and is then modified using the total ratings (TR) of 14 processing complexity characteristics to account for the different kinds of system requirements and development environments. In this model the effort can be estimated as follows:

Step1: Determine the number of components (EI, EO, EQ, ILF, and ELF)

1. **EI** - The number of external inputs. These are elementary processes in which derived data passes across the boundary from outside to inside.
2. **EO** - The number of external output. These are elementary processes in which derived data passes across the boundary from inside to outside.
3. **EQ** - The number of external queries. These are elementary processes with both input and output components that result in data retrieval from one or more internal logical files and external interface files.
4. **ILF** - The number of internal log files. These are user identifiable groups of logically related data that resides entirely within the applications boundary that are maintained through external inputs.
5. **ELF** - The number of external log files. These are user identifiable groups of logically related data that are used for reference purposes only, and which reside entirely outside the system.

Step 2: Compute the Unadjusted Function Point Count (UFC)

1. Rate each component as **low**, **average**, or **high**.
2. For transactions (**EI**, **EO**, and **EQ**), the rating is based on the **FTR** and **DET**.

2.1. **FTR** - The number of files updated or referenced.
2.2. **DET** - The number of user-recognizable fields.
2.3. Based on the table below, an **EI** that references 2 files and 10 data elements would be ranked as **average**.

FTR's	DET's		
	1 – 5	6 – 15	> 15
0 – 1	Low	Low	Average
2 – 3	Low	Average	High
> 3	Average	High	High

3. For files (**ILF** and **ELF**), the rating is based on the **RET** and **DET**.
3.1. **RET** - The number of user-recognizable data elements in an **ILF** or **ELF**.
3.2. **DET** - The number of user-recognizable fields.
3.3. Based on the table below, an **ILF** that contains 10 data elements and 5 fields would be ranked as **high**.

RET's	DET's		
	1 - 5	6 – 15	> 15
1	Low	Low	Average
2 – 5	Low	Average	High
> 5	Average	High	High

4. Convert ratings into **UFC's**.

Rating	Values				
	EO	EQ	EI	ILF	ELF
Low	4	3	3	7	5
Average	5	4	4	10	7
High	6	5	6	15	10

Step 3: Compute the Final Function Point Count (FPC)

1. Compute value adjustment factor (**VAF**) based on 14 general system characteristics (**GSC**).

General System Characteristic		Brief Description
GSC 1	Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
GSC 2	Distributed data processing	How are distributed data and processing functions handled?
GSC 3	Performance	Was response time or throughput required by the user?
GSC 4	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
GSC 5	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?
GSC 6	On-Line data entry	What percentage of the information is entered On-Line?
GSC 7	End-user efficiency	Was the application designed for end-user efficiency?
GSC 8	On-Line update	How many ILF's are updated by On-Line transaction?
GSC 9	Complex processing	Does the application have extensive logical or mathematical processing?
GSC 10	Reusability	Was the application developed to meet one or many user's needs?
GSC 11	Installation ease	How difficult is conversion and installation?
GSC 12	Operational ease	How effective and/or automated are start-up, back-up, and recovery procedures?
GSC 13	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
GSC 14	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

2. Weight each **GSC** on a scale of 0 to 5 based on whether it has no influence to strong influence.
3. Compute the **FPC** as follows.
FPC = **UFC** * (0.65 + (sum (**GSC**) * .01))

A simple linear regression can be used to estimate person-months as a function of function points [5]. The function point approach has also been adapted to generate new models. For example, the ESTIMACS model uses a modified function point method for a size estimate that is subsequently adjusted by assumptions about project complexity [33]. Another model derived from the function point approach is the

SPOR/100 model [34]. This model includes 175 product and process-related variables, although any specific estimate typically uses between 50 and 100 variables.

Although the function point approach can provide an early estimate of size, it also has certain problems. The two dominant problems associated with this metric involve the effort required to collect function point data and the difficulty in obtaining consistent estimates from multiple individuals (Kemerer, 1989).

4. Conclusion and Future work

A number of different models and effort estimation methods have been developed in the past four decades. This clearly indicates the awareness among the researchers to improve effort estimation in software engineering. There are many factors have impact on the software development process. These factors can be human, technical and their impact can never be fully predicted. We have studied the different estimation techniques and illustrated two approaches for measuring the size in the estimation process in this paper. If the estimation is done accurately, it results in error decrease. The Estimation process reflects the reality of project's progress. It avoids cost/budget or schedule overruns. This process is quite simple which takes a few inputs. This assessment framework helps inexperienced team to improve project tracking and estimation. The effort estimation still remains unreliable. Too many techniques are developed including use case point, story point etc. to overcome this inability. In the future work, we can compare these techniques for their ability.

References

[1]. Albrecht, A.J. Measuring application development productivity. In: SHARE/GUIDE: Proceedings of the IBM Applications Development Symposium, (October 1979) 83-92.
[2]. Boehm, B. Software Engineering Economics. Englewood CI@, NJ Prentice-Hall (198 1).
[3]. Putnam, L.H. A general empirical solution to the macro software sizing and estimation problem. IEEE Transactions on Sofiware Engineering, ~01.4, no. 4 ('July 1978) 345-381.
[4]. Kemerer, C.F. An empirical validation of software cost estimation models. Communications of the ACM vol. 30, no. 5 (May 1987) 416-429.
[5]. Albrecht & Gafney. Software function, source lines of code, and development effort prediction: a software science validation, IEEE Transactions on Software Engineering 9, 6 (1983) 639-648.
[6]. Shepperd, M., Schofield, C. Estimating software project effort using analogies. IEEE Transactions on Software Engineering, vol. 23, no. 12 (November 1997) 736-743.
[7]. Mukhopadhyay, T., Vicinanza, S.S., Prietula, M.J. Examining the feasibility of a case-based reasoning model for software effort estimation. MIS Quarterly (June 1992) 155-171.
[8]. *Srinivasan, K., Fisher, D. Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering, VOI. 21, no. 2 (February 1995) 126-137.*
[9]. Briand, L.C., Basili, V.R., Thomas, W.M. A pattern recognition approach for software engineering data analysis. IEEE Transactions on Software Engineering, vol. 18, no. II (1992) 931-942
[10]. *Jorgensen, M. Experience with the accuracy of Software Maintenance Task Effort Prediction Models. IEEE Transactions on Software Engineering, vol. 2 1, no.8 (August, 1995) 674-68 1.*
[11]. B.A. Kitchenham and K. Kansala, "Inter-Item Correlations among Function Points," Proc. First Int'l Symp. Software Metrics, Baltimore, Md.: IEEE CS Press, 1993.
[12]. N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Using Neural Networks in Reliability Prediction," IEEE Software, vol. 9, no. 4, pp. 53-59, 1992.
[13]. G.E. Wittig and G.R. Finnie, "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development effort," Australian J. Information Systems, vol. 1, no. 2, pp. 87-94, 1994.
[14]. F.Brooks, The Mythical Man-Month; Essays on Software Engineering, 1975. Addison-Wesley, Reading, Massachusetts.
[15]. Robert C. Tausworthe, 1981. Deep Space Network Estimation Model, Jet Propulsion Report.
[16]. Charles Symons 1991. Software Sizing and Estimation Mark II function Points (Function Point Analysis), Wiley 1991.
[17]. Allan J. Albrecht May 1984. AD/M Productivity Measurement and Estimation Validation, IBM Corporate Information Systems. IBM Corp.
[18]. Barry W. Boehm, Bradford dark, Ellis Horowitz, Chris Westland, Ray Madachy and Richard Selby. Cost Models for Future Software Lifecycle Processes: COCOMO 2.0 Annals of Software Engineering. Volume 1, pp, 57-94, 1995.
[19]. Banker, R. D., H. Chang, et al. (1994). "Evidence on economies of scale in software development." Information and Software Technology 36(5): 275-282.
[20]. Cockcroft, S. (1996). "Estimating CASE development size from outline specifications." Information and Software Technology 38(6): 391-399.
[21]. Chatzoglou, P. D. and L. A. Macaulay (1998). "A rule- based approach to developing software developmentprediction models." Automated Software Engineering 5(2): 211-243.
[22]. Dolado, J. J. (2000). "A validation of the component- based method for software size estimation." IEEE Transactions on Software Engineering 26(10): 1006- 1021
[23]. Ali Idri, Alain Abran, Taghi M. Khosgoftaar. 2001. Fuzzy Analogy- A New Approach for Software Cost Estimation. International Workshop on Software Measurement (IWSM'01).
[24]. Magne Jorgensen, A Review of Studies on Expert Estimation of Software Development Effort, March 2002.
[25]. Yunsik Ahn, Jungseok Suh, Seungryeol Kim and Hyunsoo Kim. July 2002. Journal of Software Maintenance and Evolution: Research and Practice.
[26]. Magne Jorgensen. May 2007 Forecasting of Software Development Work Effort: Evidence on Expert Judgment and Formal Model.
[27]. Parvinder S. Sandhu, Porush Bassi, and Amanpreet Singh Brar. 2008. Software Effort Estimation Using Soft

Computing Techniques. World Academy of Science, Engineering and Technology 46 2008.
[28]. Barbara Kitchenham, Emilia Mendes. 2009. Why Comparative Effort Prediction Studies may be Invalid © ACM 2009 ISBN: 978-1-60558-634-2.
[29]. Vahid Khatibi, Dayang N. A. Jawawi. 2010. Software Cost Estimation Methods: A Review. Journal of Emerging Trends in Computing and Information Science.
[30]. M. V. Deshpande, S. G. Bhirud. August 2010. Analysis of Combining Software Estimation Techniques. International Journal of Computer Applications (0975 – 8887)
[31]. Samaresh Mishra1, Kabita Hazra2, and Rajib Mall3. October 2011. A Survey of Metrics for Software Development Effort Estimation. International Journal of Research and Reviews in Computer Science (IJRRCS)
[32]. Jovan Popović1 and Dragan Bojić1. 2012. A Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle. ComSIS Vol. 9, No. 1, January 2012
[33]. Rubin, H.A. "Macroestimation of Software Development Parameters: The ESTIMACS System," IEEE Conference on Software Development Tools, Techniques and Alternatives, Arlington, VA, 1983, pp. 109-118.
[34]. Jones, C. Programming Productivity, McGrawHill, New York, NY, 1986b.