

A Review on Effort Estimation Techniques used in Software Projects

Yogesh Kumar

Pre-Ph.D. Student, CSE Deptt., UIET, MDU, Rohtak
yogs_crsce@yahoo.com

Abstract

Estimation is an important part of the project management which influences the success of project. Inaccurate and unreliable estimation can result into failure of project. As we know that software effort estimation is not an easy task for software developer, while any project starts software developer try to know how much effort are needed to complete the project. Before effort estimation first of all determine the size and complexity of software project. Complexity of software project is directly proportional to project cost and schedule. Software industries significantly differ from traditional manufacturing in numerous ways, the development effort of software products originate mainly from human resource. In this paper a review is given on the previous work done in the field of effort estimation for software Projects.

Keyword: *Software Development Effort Estimation, LOC, FP, COCOMO.*

1. Introduction

Management in any project starts with estimation. An effective estimation is the back bone for the development of any project. Without effective estimates proper project planning and tracking is impossible. During development of software project, one of the most important issues encountered is a company's ability to accurately estimate the efforts and necessary costs of the initial stage of development. Effort estimation has prompted much research during the past 30 years. However, despite the past efforts, factors that influence the development efforts and costs and the relationships among these factors have not been understood and defined properly. Thus, an adequate model for effort and cost estimation has not yet been introduced.

The models that have been established and are in use today indirectly estimate the cost and efforts by the size of the software. Among these indirect methods, line of code (LOC)-based Boehm's constructive cost model [1,2] (COCOMO) (Boehm, 1981, 1984) and function point analysis (FPA) based models [3] (Albrecht, 1979; Albrecht & Gaffney, 1983) have been used widely. The estimation models are based on statistical regression techniques. The statistical regression models estimate the software development effort by utilizing various factors, such as length of development, software complexity, types of programming languages used, etc. However,

these models fail to reflect the various development environments such as the programming language, hardware, software engineering methodology, transmission process and network technologies [4] (Venkatachalam, 1993). Therefore, there is a need to establish a new methodology for development effort estimation, which takes into account the emergence of new development environments, such as the increase in the complexity of software and information systems as well as the increase in development convenience triggered by the utilization of various package products. In order to handle the dynamic development environments, many researchers have applied the artificial intelligence (AI) based models, such as neural network models, fuzzy logic models, and case-based reasoning models, for software effort estimation. Furthermore, they have empirically proved that AI based models outperform traditional statistical regression models [5,6] (Heiat, 2002; Jun & Lee, 2001; Mukhopadhyay, Vicinanza, & Prietula, 1992; Wittig & Finnie, 1997). Due to the rapid increase in software development costs, software development imposes a heavy burden on companies seeking to implement enterprise information systems. Before performing the costly software developing processes, it is necessary to examine and evaluate the anticipated costs and profits of such projects. The estimation accuracy of software development cost plays an important role, both directly and indirectly, in a company's decision on whether to make an investment or not. Ongoing attempts exist to evaluate and measure the cost required for software development and the most prominent method indirectly estimates the necessary cost and efforts by the size of the software. There have been various studies concerning such estimation approaches [7]. Myrtveit and Stensrud (2005) distinguish between sparse data methods and many-data (data-driven) methods.

Sparse-data methods are estimation methods requiring few or no historical data. They include knowledge-based and case-based reasoning models. Under the assumption that estimations by human experts are better than all data-driven models, knowledge-based model uses human expertise, rather than previous project data, to generate predictions. However, compared with data-driven models, it has limited ability in handling quantitative estimation of software effort (Jun & Lee, 2001). Compared with sparse-data methods, many-data methods

build estimation models based on historical data. Statistical models and neural network models are typical examples in this category. Among statistical methods, Boehm's COCOMO model based on LOC (Boehm, 1981, 1984) and Albrecht and Gaffney's model based on FPs [3] (Albrecht & Gaffney, 1983) are frequently used. Since two types of effort estimation models are mainly based on simple linear regression models, they have very limited capabilities in terms of reflecting complex software development environments. As the prediction power of neural networks under dynamic and uncertain environment has been widely recognized, many researchers have attempted to utilize neural networks in estimating software development efforts. However, they have evaluated the neural network models without first considering the nature of the data themselves.

Shepperd and Kadoda [8] state that the underlying data characteristics, such as size, number of features, and type of distribution, strongly influence the performance of estimation models. In developing NN based models, it is important to look first at the organization of the data to remove outliers and identify dominant variables.

The Chaos report conducted by the Standish Group [10], [2], which is the report on the failure of projects in the field of information technologies measures the success of projects only if completed in time, within budget, and if they met the requirements. Several papers in software development effort estimation have cited the Chaos report [11,12,13,14,15],[3][4][5][6][7]. Recently this report found that more than half of software projects (around 63%) conducted between 2002 and 2010 were delivered with delay, were over budget and many were not even finished; just 37 percent were classified as successful. The main cause of these problems is a failure of the software development effort estimation [16, 17], [8][9].

2. Effort Estimation Techniques

Generally there are many methods and techniques for software cost estimation which are divided into two groups [53]:

1. Non Algorithmic
2. Algorithmic

In the Algorithmic method we use a formula for calculating the cost estimation. The formula is created by combining related cost factors in the various models. Non-algorithmic methods do not use any formula to calculate the software cost estimation. Both groups are useful for performing the accurate estimation. If the requirements are known better, their performance will be better.

2.1 Non Algorithmic

(i) Expert Judgment

In the Expert judgment technique we can estimate by getting advice from different expert who have an extensive experiences in similar projects. This method is usually used when there is limitation in finding data and gathering the requirements. Consultation is the basic issue in this method. One of the most common methods which work according to this technique is Delphi. Delphi consists of group of expert and coordinator and conducts an especial meeting among the project experts and tries to achieve the true information about the project from their debates. Delphi includes some steps:

- a) The coordinator provides an estimation form to each expert.
- b) Each expert completes their individual estimate anonymously (without discussing with others).
- c) The coordinator gathers all forms and sums up them (including mean or median) on a form and asks experts to start iteration.
- d) Coordinator prepares and distributes a summary of the response of all the estimators in an iteration form.
- e) Steps (b-d) are repeated until an approval is gained.

(ii) Estimation By Analogy

In this method, several similar completed software projects are analyzed and estimation of effort and cost are done according to their actual cost and effort. Estimation by this technique is accomplished at the total system levels and subsystem levels. By assessing the results of previous actual projects, we can estimate the cost and effort of a similar project. The steps of this method are considered as:

- a) Choosing of analogy.
- b) Investigating similarities and differences.
- c) Examining of analogy quality.
- d) Providing the estimation.

2.2 Algorithmic

Algorithmic method use formula and provide some mathematical equations to perform software estimation. These mathematical equations are based on historical data and use inputs such as Source Lines of Code (SLOC), number of functions to perform, and other cost drivers such as language, design methodology etc.

(i) LOC Based

LOC is the simplest among all metrics available to estimate project size. This method measures the size of a project by counting the number of source instructions in the developed program. While counting the number of source instructions; lines used for commenting the code and the header lines are ignored. Determining the LOC count at the end of the project is very simple but accurate estimation of LOC at the start of the project is very difficult. Accurate estimation of size acts as a base for other estimation of project activities and any wrong estimation lead to wrong estimates of other dependent activities. In order to estimate the LOC count in the beginning of a project, we have to make a systematic guess. Every project cannot be taken as a single problem; it is divided into several sub problems and further in modules by project managers until the size of deepest level module can be predicted. To make our estimation precise we can use the past experience of similar projects. After predicting and estimating lowest level module size, the total project size can be estimated by adding estimates level by level.

(ii) Function Point

[54]At first, Albrecht (1983) presented Function Point metric to measure the functionality of project. In this method, estimation is done by determination of below indicators:

- Number of User Inputs,
- Number of User Outputs,
- Number of Logic files,
- Number of Inquiries,
- Number of Interfaces

A Complexity Degree which is between 1 and 3 is defined for each indicator. 1, 2 and 3 stand for simple, medium and complex degree respectively. Also, it is necessary to define a weight for each indicator which can be between 3 and 15.

At first, the number of each mentioned indicator should be tallied and then complexity degree and weight are multiplied by each other. Generally, the unadjusted function point count is defined as below:

$$UFC = \sum_{i=1}^5 \sum_{j=1}^3 N_{ij} W_{ij}$$

where N_{ij} is the number of indicator i with complexity j and; W_{ij} is the weight of indicator i with complexity j . According to the previous experiences, function point could be useful for software estimations because it could be computed based on requirement

specification in the early stages of project. To compute the FP, UFC should be multiplied by a Technical Complexity Factor (TCF). Each component can change from 0 to 5. 0 and 5 indicate that the component has no effect on the project and the component has strong effect and very important respectively. Finally, the TCF is calculated as:

$$TCF = 0.65 + 0.01(\text{SUM}(F_i))$$

The range of TCF is between 0.65 (if all F_i are 0) and 1.35 (if all F_i are 5). Ultimately, Function Point is computed as

$$FP = UFC * TCF$$

(iii) Cocomo Model

Model-1(Basic COCOMO Model):-

[55]The basic COCOMO model is a static model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC). This model has three development modes that is organic, semidetached and embedded. To determine the initial effort in person-months the equation used is of the type

$$\text{EFFORT} = x * (\text{KLOC})^y$$

$$\text{TDEV} = 2.5 * (\text{EFFORT})^z$$

TDEV is the estimated time to develop the software expressed in month. The value of constants x , y and z depend on the project type. It has following three classes of software projects describe in table I.

Table – I

Basic COCOMO	x	y	z
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Model2(Intermediate COCOMO Model):-

Intermediate COCOMO Model computes software development effort as a function of program size and set of cost drivers that include subjective assessment of the products, computer, personnel and project attributes.

The same basis equation for the intermediate cocomo model is used, but it include fifteen cost drivers are rated on a scale of very low to very high that is used to calculate the effort multiplier and each of them returns an adjustment factor which multiplied yields in the total EAF (Effort Adjustment Factor). The adjustment

factor is 1 is consider as normal. The intermediate COCOMO model takes the form:

$$\text{EFFORT} = x * (\text{KLOC})^y * \text{EAF}$$

$$\text{TDEV} = 2.5(\text{EFFORT})^z$$

In addition to the EAF, the model parameter "x" is slightly different in Intermediate COCOMO from the basic model. The parameter "y" remains the same in both models.

Table II

Intermediate COCOMO	x	y	z
Organic	3.2	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	2.8	1.20	0.32

Model 3 (Detailed COCOMO Model):-

The detailed COCOMO Model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc) of the software engineering process.

(iv) Putman's model

This model has been proposed by Putman according to manpower distribution and the examination of many software projects (Kemerer,2008). The main equation for Putnam's model is:

$$S = E x (\text{Effort})^{1/3} T_d^{4/3}$$

where, E is the environment indicator and demonstrates the environment ability. T_d is the time of delivery. Effort and S are expressed by person-year and line of code respectively. Putnam presented another formula for Effort as follows:

$$\text{Effort} = D_0 x T_d^3$$

where, D_0 , the manpower build-up factor, varies from 8(new software) to 27(rebuilt software).

3. Literature Review

Barry Boehm [1] aims at estimating the development effort from famous COCOMO - I, for small to medium sized software projects. The value of development effort

depends on size of software and empirically determined constants. Further, Barry Boehm et. al. [24], [2] also discusses the revised version of COCOMO-I as COCOMO-II that uses some fixed values for one constant and the other constant depends on the scaling factors. Yinhuang Zheng et.al. [25], [3] computes development effort for programming measurement and estimation on the basis of a constant 5.5, as multiplier for the development effort. Stephen Mc Donnell [26], [4] estimates the development effort on the basis of data collected from organization that captures environmental factors and also considers differences among the given projects. Albrecht and Gaffney [27], [5] discusses about function point as a unit of measurement to express the software functionality. Matson et. al. [28], [6] uses function point or the software cost estimation with a very high constant factor i.e. 585.7 for the calculation of development effort. Clemmons R.K. [29], [7] discusses the functional scope of the project by analyzing the contents on the basis of use case measures Magne Jorgenson [30], [8] attempts to provide six criteria for evaluation of the development methods like - automation, comprehensive assessment, objectivity, specification, testing and validity for the software to be developed. Bernard et. al. [31], [9] discusses the result of two case studies and two experiments to show the impact of effort estimates on software project. It also works on preplanning of effort estimation and proposes that too low effort estimates lead to less effort and more errors compared with more realistic effort estimates. Noureldin AZ Adem and Zarinah M [32], [10] considers both functional and non functional requirements and discusses about an automated tool to estimate the size of software projects on the basis of two processes namely - goal and scenario based requirements elicitation technique and text based function point extraction guidance rules. IEEE Computer Society [33], [11] explains the IEEE recommended practices for the correct and appropriate way of writing software requirement specification (SRS) document. This is IEEE 830:1998 standard. Geoffrey K Gill and Chris F. Kemerer [34], [12] discusses about complexity density ratio as a useful predictor for software maintenance productivity that is derived on the basis of cyclometric complexity. Charles R Symon [35], [13] discusses about function points analysis (FPA) and compares the original FPA with FP mark-II, as an improvement. Sharma Ashish and Kushwaha D.S. [37,38], [15,16] proposes and establishes the improved requirement based complexity (IRBC) measure obtained on the basis of SRS of the proposed software. Maurice J Halstead [39], [17] discuss about a measure based on the principle of count of operators and operands and their respective occurrences in the code for the estimation of difficulty, effort and time. Kushwaha D.S. and Misra A.K. [40], [18] discuss CICM and modeling of test effort based on component of the shelf (COTS) & component based software engineering (CBSE). The [41], [19] discusses about importance of defining the user requirements and their impact on software development

process.

Defect predictors have used size and complexity metrics when attempting to predict defects of software. The earliest study along this line is the one reported by Akiyama [42], which dealt with the system developed at Fujitsu, Japan. Software metrics have been used as quantitative means of assessing software development process as well as the quality of software products. Many researchers have studied the correlation between software design metrics and the likelihood of occurrence of software faults [44,45,47,48,49,50].

As the size and complexity of software systems increases, software industry is highly challenged to deliver high quality, reliable software on time and within budget. Although there is a diversity of definitions of software quality, it is widely accepted that a project with many defects lacks quality. Defects are commonly defined as deviations from specifications or expectations that might lead to future failures in operation [43,46]. Knowing the causes of possible defects as well as identifying general software management decisions and practices that may need attention since the beginning of a project could save money, time and work. The estimation of the potential fault-prone of software is an indicator of quality and can help planning, controlling and executing software development and maintenance activities [43].

4. Conclusion

This paper provides an overview of different techniques on software effort estimation and review of research done in last 10 years on effort estimation. The estimation processes of the development effort consist of specific activities:

- 1) Obtaining data from previous projects.
- 2) Generation of estimation models.
- 3) Verifying and validating the models, based on accuracy.

For accurate effort estimation 3P principle must be accurately estimated. i.e. People, Product and Process. Means including factors like: number of skilled person needed, Project Quality, Task complexity etc. Lot of effort estimation techniques have been proposed and researched over the last years [21], [14]. Researchers aimed at (1) determining which technique had the greatest effort prediction accuracy, or (2) proposing new or combined techniques that could provide better estimates.

References

- [1] Boehm, B. W. (1981), Software engineering economics. Prentice Hall.
- [2] Boehm, B. W. (1984), Software engineering economics. IEEE Transactions on Software Engineering, 10(1).
- [3] Albrecht, A. J., & Gaffney, J. E. (1983), "Software function, source line of code and

development effort prediction: A software science validation", IEEE Transactions on Software Engineering, SE-9(6), 639– 648.

- [4] Venkatachalam, A. R., (1993) "Software cost estimation using artificial neural networks", In Proceedings of international joint conference on neural networks (pp. 987–990).
- [5] Heiat, A. (2002), "Comparison of artificial neural networks and regression models for estimating software development effort", Information and Software Technology, 44, 911–922.
- [6] Mukhopadhyay, V. P., Vicinanza, S. S., & Prietula, M. J. (1992) "Examining the feasibility of a case-based reasoning model for software effort estimation", MIS Quarterly, 16(2), 155–171.
- [7] Myrtveit, I., & Stensrud, E. (2005) "Reliability and validity in comparative studies of software prediction models", IEEE Transactions on Software Engineering, 31(5), 380–391.
- [8] Shepperd, M., & Kadoda, G. (2001), "Comparing software prediction techniques using simulation", IEEE Transactions on Software Engineering, 27(11), 1014–1021.
- [9] SWEBOK executive editors, Alain Abran, James W. Moore ; editors, Pierre Bourque, Robert Dupuis. (2004). Pierre Bourque and Robert Dupuis. ed. Guide to the Software Engineering Body of Knowledge - 2004 Version. IEEE Computer Society. pp. 1–1. ISBN 0-7695-2330-7.
- [10] Standishgroup.com. 2012, The Standish Group ©. www.standishgroup.com
- [11] De Araújo R.A., Soares S., Oliveira L.I. 2012a. Hybrid morphological methodology for software development cost estimation. Expert Systems with Applications. 6129-6139.
- [12] Bonneti A., Bortot S., Fedrizzi M., Marques Pererira R.A., Molinari A., 2012. Modelling group process and effort estimation in project management using the Choquet integral: An MCDM approach. Expert Systems with Applications.
- [13] De Araújo R.A., Oliveira A.L.I., Soares S., Meira S., 2012b. An evolutionary morphological approach for software development cost estimation. Neural Networks. 285-291.
- [14] Lagerström R., Von Würtemberg L. M. Holm H., Luczak O., 2012. "Identifying factors affecting software development cost and productivity", Software Quality Journal. 395-417. DOI 10.1007/s11219-011-9137-8.

- [15] Moløkken-Østfold K., Jørgensen M., 2003, “A review of surveys on software effort estimation”, in International Symposium on Empirical Software Engineering (ISESE 2003), IEEE Computer Society, Rome, Italy. Pp. 223–230.
- [16] De Araújo, A. R., de Oliveira, A. L. I., & Soares, S. C. B. 2009, “A morphological-rank linear approach for software development cost estimation”, In IEEE international conference on tools with artificial intelligence. IEEE.
- [17] De Araújo, R. A., Oliveira, A. L. I., & Soares, S. C. B. 2011, A shift-invariant morphological system for software development cost estimation. *Expert Systems with Applications*, 38(4), 4162–4168.
- [18] Rombach D., Münch J., Ocampo A., Humphrey W. S., and Burton D., 2008 “Teaching disciplined software development”, *Journal Systems and Software*, Elsevier, pp. 747-763.
- [19] Naur P. and Randell B., 1969, *Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*.
- [20] Brooks F. P. Jr., January 2003. “Three Great Challenges for Half-Century-Old Computer Science”, *Journal of the ACM*, Vol. 50, No.1, pp. 25-26.
- [21] Jørgensen M., Shepperd M., 2007, “A systematic review of software development cost estimation. Studies”, *IEEE Transactions on Engineering Management* 33(1).
- [22] Lopez-Martin C., Isaza C., Chavoya A., 2012, “Software development effort prediction of industrial projects applying a general regression neural network”, *Empir Software Eng.* Pp 738–756. DOI 10.1007/s10664-011-9192-6.
- [23] Boehm B. and Abts C., 2000. *Software Development Cost Estimation Approaches – A Survey*. University of Southern California. Los Angeles, CA.
- [24] B.W. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall, 1981 B.Boehm, E Horowitz, R.Madachy, B.Clark, C.Westland, R.Selby, “Cost models for future software lifecycle process:COCOMO 2.0”, *IEEE computer society*, pp 1-31, 1987
- [25] Yinhan Zheng et.al. “Estimation of the software project effort based on function point”,*IEEE-ICCSE-2009*
- [26] Stephen Mac Donell, “Comparative review of functional complexity assessment methods for effort estimation”, *Software Engineering Journal* pp. 107-116, 1994.
- [27] Alan J Albrecht and John E Gaffiney, “Software functions, source lines of code and development effort prediction: A software science validation”, *IEEE Transactions on Software Engineering*, Vol SE-9, No.6, pp- 639-648, November 1983
- [28] Matson, J.E.; Barrett, B.E.; Mellichamp, J.M., “Software Development Cost Estimation using function points”, *IEEE Transaction of Software Engineering*, Vol. 20, Issue 4, pp-275 – 287, 1994
- [29] Clemmons. R.K., “Project estimation with Use Case Points”, *The Journal of Defense Software Engineering*. pp. 18-22,February, 2006
- [30] Magne Jorgensen, Dag IK Solberg, “Impact of effort estimate on software project work”, Elsevier, *Info & Soft. Tech.* pp. 939-948, 2006.
- [31] Bernhard Peschi et. al., “Recommending effort estimate method for software project management”, *IEEE-MC-ACM*, pp. 77-80,2001.
- [32] Noureldin AZ Adem, Zarinah M Kasirun, “Automating function point analysis based on the functional and the non-functional requirement text”, pp. 664-669, *IEEE Conf.* 2010.
- [33] IEEE Computer Society, *IEEE recommended practice for software requirement specification*, IEEE std 830-199835.
- [34] Geoffrey K. Gill, Chris F. Kemmerer, “Cyclometric complexity density and software maintenance productivity”, *IEEE transactions on software engineering* vol.17,no.12, pp. 1284-1288, December 1991.
- [35] Charles R Symons, “Function point: Difficulties and Improvements”, *IEEE Transactions on Software Engineering*, Vol.14,No.1, pp. 2-11, January 1988
- [36] Sharma Ashish, Kushwaha D.S., “NLP/ POS tagger based requirement analysis and its complexity analysis”, *ACM SigSoft* ,Vol.36, Issue 2, pp. 1-14, January 2011
- [37] Sharma Ashish, Kushwaha D.S., “Complexity measure based on requirement engineering document and its validation”, *IEEE Conf. on Computer and*

- Communication Technology , pp. 608-615, ICCCT 2010
- [38] M.H. Halstead, Elements of Software Science, New-York: Elsevier, 1977
- [39] DS Kushwaha, AK Misra, “Software Test Effort Estimation”, ACM Sig Soft, Software Engineering Notes, Vol. 33 No. pp. 1-6,3, May 2008
- [40] The Standish group research for staggering bugs and effort, <http://standishgroup.com>
- [41] F. Akiyama, An example of software system debugging, Information Processing 71 (1971) 353– 379.
- [42] S. Bibi, G. Tsoumakas, I. Stamelos, I. Vlahavas, “Regression via classification applied on software defect estimation”, Expert Systems with Applications 34 (2008) 2091–2101.
- [43] L.C. Briand, W.L. Melo, J. Wust, “Assessing the applicability of fault-proneness models across object-oriented software projects”, IEEE Transactions on Software Engineering 28 (2002) 706–720.
- [44] K.E. Emam, W. Melo, C.M. Javam, “The prediction of faulty classes using object-oriented design metrics”, Journal of Systems and Software 56 (1) (2001) 63– 75.
- [45] N.E. Fenton, M. Neil, “A critique of software defect prediction models”, IEEE Transactions on Software Engineering 25 (5) (1999) 675–689.
- [46] T.L. Graves, A.F. Karr, J.S. Marron, H. Siy, “Predicting fault incidence using software change history”, IEEE Transactions on Software Engineering 26 (7) (2000) 653–661.
- [47] A. James, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, “Identification of defect-prone classes in telecommunication software systems using design metrics”, Information Sciences 176 (2006) 3711–3734.
- [48] S.-K. Oh, W. Pedrycz, B.-J. Park, “Self-organizing neurofuzzy networks in modeling software data”, Fuzzy Sets and Systems 145 (2004) 165–181.
- [49] T.-S. Quah, M.M.T. Thwin, “Prediction of software development faults in PL/SQL files using neural network models”, Information and Software Technology 46 (2004) 519–523.
- [50] Rambaugh., A Theory of Object-Oriented Design: The building-blocks of OOD and notations for representing them (with focus on design patterns.)
- [51] Ashok Kumar and Anil Kumar, “Complexity Measurement During Reengineering by Using Pattern Based Metrics”, International Journal of Research and Reviews in Computer Science (IJRRCS) Vol. 2, No. 6, December 2011, ISSN: 2079-2557
- [52] Josephin.M.Baby, Anusha M.E , Rajeswari Mukesh , “A Study On Software Cost Estimation”, international journal of emerging trends and Technology in computer science(IJETTCS) special issue ISSN 2278-6856,2013.
- [53] Vahid Khatibi, Dayang N. A. Jawawi, “Software Cost Estimation Method:A Review”, Journal of emerging trends in computing and information sciences, Volume 2 No. 1.
- [54] Magne J and Martin S, “A Systematic Review of Software Development Cost Estimation Studies”, IEEE Transactions On Software Engineering, Vol. 33, No. 1, pp. 33-53, January 2007.