ANALYSIS OF SIZE METRICS AND EFFORT PERFORMANCE CRITERION IN SOFTWARE COST ESTIMATION

S.Malathi

Research Scholar, Department of Computer Science,
Sathyabama University, Chennai.
Tamilnadu, India.
malathi_raghu@hotmail.com

Dr.S.Sridhar

Research Supervisor, Department of CSE & IT,
Sathyabama University, Chennai.
Tamilnadu, India
dakyesyes@gmail.com

Abstract

Effective management of any software process requires quantification measurement and modeling. Software metrics provide a quantitative basis for the development and validation of methods utilized in software cost estimation process and can also be used to improve the productivity and quality of the process. During initial stages of software cycle, it is imperative for the project managers to recognize the merits and de-merits of the metrics and use the appropriate metrics in the estimation process. Software size and effort performance metrics continues to be a controversial issue in the software engineering environment. The paper gives an overview of the metrics that are used for software size and effort performance by the software estimation community. The metrics that are in vogue are inadequate to achieve optimum results in estimation. The present analysis depicts that the prevailing metrics are not applicable for diverse techniques. The results are bound to improve by continual analysis with various metrics and techniques.

***Keywords*:** software metrics; quality; size metrics; performance metrics; cost estimation.

1. Introduction

Software metrics plays a pivotal role in the planning and control of software development projects. It has shown the right path to the software community in deciding the reliable method to assign the correct value to software solutions, thereby enabling them to evaluate the impact of the detours and missteps of the metrics during the course of work. Metrics [1] are derived from the earlier data and used to compare the current state of the program with past performance or prior estimates. Metrics depict trends of increasing or decreasing values, which are comparable with the previous value of the same metric. It is noted that software program might consume too many resources if errors made in the requirements phase were not discovered and corrected before the coding phase.

Measurement must be integrated into the total software life cycle. To be effective, metrics must not only be collected but should be used. A good measurement program facilitates early detection of problems and provides quantitative clarification of critical development issues leading to well defined success of the project [2]. Metrics give ability to the software community to identify [3], resolve, and/or curtail risk issues before they surface.

Size measure is perhaps the most commonly used metrics because it is a good indicator of memory requirements, effort and development time. Poor size estimation is one of the main reasons for the ultimate failure of major software-intensive acquisition programs. Size is the vital factor in determining the cost, schedule, and effort of estimation. The failure to accurately predict (usually too small) the expected results in budget will escalate the cost and increase the time schedule which will undermine the confidence and erode support for the program. Size estimation is complex and its results should be constantly updated with the actual tally throughout the life cycle.

Complexity is a function of size, which greatly impacts the design errors and concealed defects resulting in quality problems, cost overruns, and schedule slips. Complexity must be continuously monitored, measured and controlled. Any impact on size metrics is reflected in the effort performance criterion. The effort metric estimates the realistic effort required to maintain a project. Effort estimates [4] may be used as an input to project plans, iteration plans, budgets, investment analyses and pricing processes.

This paper examines the various approaches used to measure software size and effort performance criterion for the last two decades. It focuses on the reasons for application of the approaches, the technological or human factor that was in play and the degree of success achieved in the use of each approach.

2. Review Process

In order to achieve the main objective of finding out the suitable metrics for proper analysis of existing techniques, the past research has been systematically and methodically scrutinized initially through manual search process of the major journals and seminar publications related to size and effort performance metrics. Papers based on the same data set but with different focus are considered as different papers. Fortunately, such type of cases was not considerable to effect significant changes in the outcome of the analysis and therefore all papers were reviewed. However, while carrying out a review of a particular phenomenon, e.g., the sturdiness of a particular finding, a clearer distinction between study paper and inclusion process is made. These are read thoroughly relating to software cost estimation and the details are tabulated in table 1 and represented in figure1.

Table 1.Distribution of Results

S.No	Titles	Percent
1	Conference papers	48%
2	Journal papers	35%
3	Thesis	4%
4	Web articles	3%
5	Books	10%

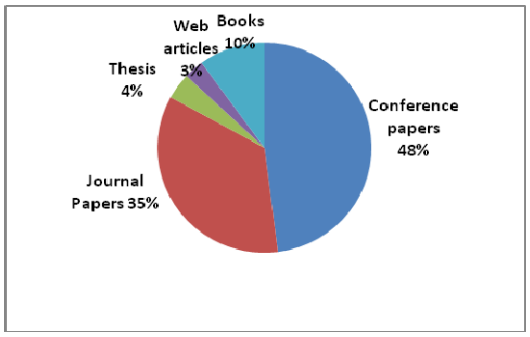

Fig.1. Types of Publications

From the figure, it is inferred that software metrics have been projected more in conferences and journals rather than other titles

3. Metrics

The various size metrics and effort performance metrics are analyzed and discussed below.

3.1. *Size Metrics*

As software development moved out from simulations to the real world, it immediately became obvious that ability is invariably essential to measure productivity and quality**.** Size measure is the simplest among all available metrics to estimate the project as it is easy to determine and form one of the most important factors influencing software development, including its productivity. The various size metrics include:

3.1.1. *Source Lines Of Code*

SLOC measure [5] is a count of the number of machine instructions developed and was the first measure applied to software. It is very easy to count the instructions at the initial stage. Usually, this metric is estimated by dividing the project into several modules and sub-modules until the size can be estimated approximately. This measure includes two types of SLOC.

- Logical SLOC
- Physical SLOC

The SLOC metric [6] is supported by various cost estimating tools and historical data. Thus, SLOC is considered as the simplest measure to come up with a number. In general, it is fair to say that SLOC, considered in a vacuum, is a poor way to measure the value that is delivered to the end user [7]. The KDLOC (1000 lines of code) can be used to estimate a complex project.

3.1.2. *Function Point Metrics*

Function point metric was proposed by Albrecht in 1979. The concept underlying in this method is that the size of a software product is directly dependent on the number of different functions or the features it supports. It is a way to measure the productivity that is independent of technology and environmental factors. The various parameters are:

- Number of inputs
- Number of user outputs
- Number of inquiries
- Number of files.
- Number of interfaces

$$FP = UFP*VAF. \quad (1)$$

The unadjusted function point (UFP) is given as

$$UFP = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{ij} W_{ij} \quad (2)$$

Where N_{ij} and W_{ij} are the number and weights of types of class i with complexity j, respectively. The value adjustment factor is calculated as

$$VAF = 0.65 + 0.01 * \sum_{i=1}^{14} C_i \quad (3)$$

Where C_i is the list of 14 general system characteristics.

Function Point being a non-code oriented size measure is used in information system. This is independent of the programming language, product design or development style. Moreover, it is best suitable for data intensive system. It is determined that information obtained by the parameters is not adequate to determine the value of the software behind the scenes.

3.1.3. *Object Point metrics*

Object points (alternatively named application points) are an alternative function-related measure to function points when 4GL [8] or similar languages are used for development.

Object points are not the same as object classes. The number of object points in a program is a weighted estimate of

- The number of separate screens that are displayed
- The number of reports that are produced by the system
- The number of program modules that must be developed to supplement the database code

Object points are an alternative for function points and good for systems using High Level Application Building Tools. It takes into account of the object-counts (i.e) number of screens, reports and program modules. It includes the complexity levels SIMPLE, MEDIUM and DIFFICULT. Thus, the estimation of an object point is easier from a high-level software specification. It doesn't depend on the implementation details and hence used at an early stage.

3.1.4. *Halstead Metrics*

In 1977, Halstead introduced complexity metric based on the number of operators and operands in a program with an analogy to verbs and nouns in a document. This measure is very simple and applicable to all programming languages. Halstead considered programs to be constructed of tokens, which could be either operators or operands. The equations for difficulty, effort and volume are called as "SOFTWARE SCIENCE".

$$n = n1+n2. \quad (4)$$

Where n is the program vocabulary and n1, n2 are the number of operators and operands. The program length is evaluated as

$$N = N1+N2 \quad (5)$$

Where N1, N2 are the total number of operators and operands. The program volume is

$$V = N (\log2 (n)) \quad (6)$$

N, n is the program length and program vocabulary. The difficulty is measured by

$$D = (n1/2) * (N2/n2) \quad (7)$$

Potential Volume

$$V^* = (2+n2^*) \log_2(2+n2^*) \quad (8)$$

Program Level

$$L= V^*/V \quad (9)$$

Where n1, n2 and N2 is the number of distinct operators, operands and total operands.

Halstead measure has its limitations as it depends on completed code and has little or no use as a predictive estimating model.
Halstead observed [9] the following:

- Code complexity increases as volume increases
- Code complexity increases as program level decreases

3.1.5. *Mc Cabe's Metric*

This measurement model was developed by Thomas J. Mc Cabe in 1976 and is used to indicate the complexity of a program. Cyclomatic complexity V (G) is the measure of program complexity of the code. However, it fails to differentiate the complexity of some rather simple cases involving single conditions in conditional statements. As indicated in size metrics, measures of complexity that can be computed early in the software cycle will be of greater value in managing the process.

$$V(G) = E-N+2 \quad (10)$$

E - no. of edges N- no. of nodes

$$V(G) = P+1 \quad (11)$$

P - Predicate node

$$V(G) = \text{closed area}+1 \quad (12)$$

This metric gives relative complexity of various designs and therefore used as a quality metric. It is measured early in the life cycle compared to Halstead and also very easy to apply. It is used to identify the best area in testing and its accuracy depends on how well the nested loops are understood.

3.1.6. *PERT Metric*

PERT is an estimating technique [1] and involves experts' judgement of three possible code-sizes: S_l, the lowest possible size; S_h the highest possible size; and S_m, the most likely size. The estimate of the code-size S is computed as:

$$S = (S_l + S_h + 4S_m)/6 \quad (13)$$

PERT can also be used for individual components to obtain an estimate of the software system by summing up the estimates of all the components. It is flexible for increasing or decreasing the project speed but not scalable for small projects.

Many metrics described here continues to be popular measures for software cost estimation. Even as function point, PERT, Halstead and McCabe metrics have emerged better in the software community and it should be noted that many of the popular methologies used for cost estimation rely on SLOC.

3.2. *Effort Performance Metrics*

The current state of software effort performance criterion is not very satisfying. In the past, many metrics and number of methods in cost estimation have been proposed. Unfortunately, most of them defined have lacked one or both of two characteristics:

- Sound conceptual, theoretical bases
- Statistically significant experimental validation

Most performance criterion metrics [10] have been defined by an individual and then tested in a very limited environment. The metrics are:

3.2.1. *Mean Magnitude Relative Error (MMRE)*

The most widely used evaluation criterion to assess the performance of software prediction is the MMRE [11] this is computed as:

$$MMRE(\%) = \frac{1}{n}\sum_{i=1}^{n} MRE * 100 \quad (14)$$

Where $MRE = | \hat{E} - E | / \hat{E}$

The advantage is that the comparison can be made across datasets and it is independent whether effort is measured in work/hours or work/months. The main drawback is that this method is not independent of scale. [12]

3.2.2. *Prediction (n)*

Prediction at level n is defined as the percentage of projects that has absolute relative error less than n. Moreover, this measure deals more with underestimation [13].

3.2.3. *Mean Absolute Relative Error (MARE)*

$$MARE(\%) = \frac{1}{n}\sum_{i=1}^{n} MRE * 100 \quad (15)$$

Where $MRE = \left| \frac{\hat{E} - E}{\hat{E}} \right|$

3.2.4. *Variance Absolute Relative Error (VARE)*

$$VARE(\%) = Var\left[\left| \frac{\hat{E} - E}{\hat{E}} \right| \right] * 100 \quad (16)$$

3.2.5. Balance Relative Error (BRE)

$$BRE = \frac{\left| \hat{E} - E \right|}{\min(\hat{E}, E)} \quad (17)$$

3.2.6 Value Accounted For (VAF)

$$VAF(\%) = \left(1 - \frac{\mathrm{var}(E - \hat{E})}{\mathrm{var}\, E} \right) * 100 \quad (18)$$

3.2.7 Median Magnitude of Relative Error (MdMRE)

MdMRE= median (MRE) (19)

3.2.8 Root Mean Square Error (RMSE)

RMSE is recurrently used measure of variation between values predicted by a model or estimator and the values actually experimental from the thing being modeled or estimated[14].It is just the square root of the mean square error which measures the average magnitude of error as shown in equation given below:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(\hat{E} - E \right)^2} \quad (20)$$

RMSE can only be compared between models whose errors are measured in the same units. There is no absolute criterion for a "good quality" value of RMSE.It depends on the units in which the variable is measured and on the degree of forecasting precision.

N=No of Projects, E= Estimated Effort, $\hat{E}$ = Actual Effort

A model which gives higher VAF is better than the one which gives lower VAF. A model which gives higher Pred (n) is better than model which gives lower Pred (n) [15]. A standard criteria for considering the model as acceptable is Pred (0.25) ≥ 0.75. This means that at least 75% of the estimate is within the range of 25% of actual values. A model which gives lower MARE is better than that which gives higher MARE [16] [17]. A model which gives lower VARE is better than the one which gives higher VARE [18]. A model which gives less MMRE [19] is better than the model which is having higher MMRE. Moreover, the MMRE or MARE can be used on the dataset even under a complex trial and error heuristic evaluation [20]. A model which gives lower BRE is better than that the model which gives higher BRE. MdMRE [21] exhibits a similar pattern to

MMRE but it is more likely to select the true model especially in the under estimation cases since it is less sensitive to extreme outliers.
The discussion on the various criterions can be summarized as:

- The difference measure is not suitable for the evaluation of the software models.
- The mean measure can take into account of every single relative error, but they are significantly influenced by outliers.
- Single value measures are not influenced by outliers and therefore are not always reliable.

The above said points warrant the need for better measures in accuracy evaluation.

4. Review Analysis

The above review on various metrics used in software size and effort estimation provides a clear understanding regarding their roles in cost estimation. From Table 2, it is concluded that the percentage of accuracy is more in terms of MMRE [22] and the sensitivity is also very high. The variations are shown in Figure 2 and Figure 3.

Table 2. Effort Performance Analysis in cost estimation

METHODS	MMRE	PRED	MdMRE	MARE	VARE	BRE	VAF
Algorithmic Methods	48%	34%	2%	8%	3%	2%	3%
Non-algorithmic Methods	44%	35%	4%	5%	4%	4%	4%

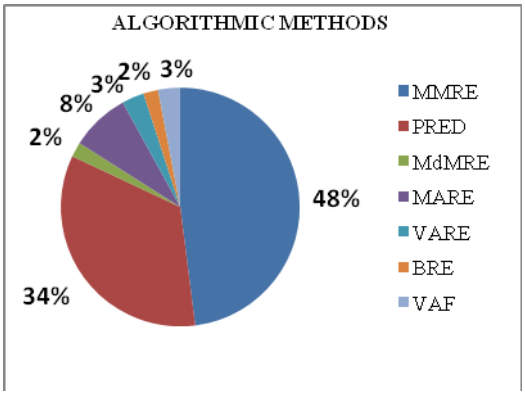

Fig.2 Performance of metrics in algorithmic models

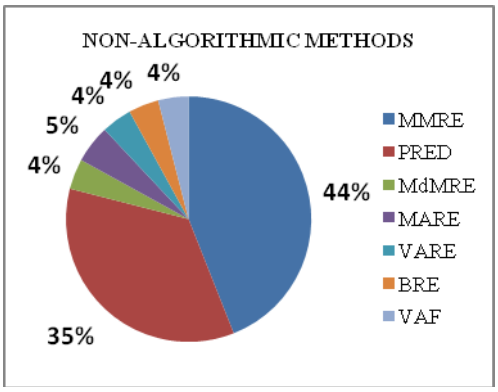

Fig.3. Performance of metrics in non algorithmic models

In some cases, significant successes have been reported in the validation. However, subsequent attempts to use the metrics in other methods have yielded very different results. Nevertheless, discrepancies and disagreements

in reported results have left many observers with the sense that the field of software metrics is, at best, insufficiently mature to be of any practical use.

5. Conclusion

In order to evaluate the models and improve the software process, we need appropriate metrics to measure and determine the model parameters. In this paper, a comprehensive overview of size metrics and performance metrics is provided and an up-to-date review of its latest performance is made to allow the team members for better understanding of the underlying methods involved and when to use these metrics. Finally, it becomes important to stop looking for a single unit for software size measure and effort performance. The scope of software metrics has multiple dimensions. The amount of user functionality is an important dimension but if viewed alone it has limited value outside. External benchmarking and productivity studies need to be performed within stratified categorizations of feature functional and non-functional complexity. Efforts are taken continuously to pursue a better measure to describe the metrics of size and effort performance criterion while at the same time, attempting to bridge the gap between the IT and the business language software.

References

[1] Fenton.N.E and Pfleeger.S.L, (1997) :Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company.

[2] Chris F. Kemerer, (1987): An Empirical Validation of Software Cost Estimation Models, Communications of the ACM, Vol. 30, No. 5,pp. 416-429.

[3] Wijayasiriwardhane.T, Lai.R, Kang.K.C, (2011): Effort estimation of component based software development-a survey , Vol.5,Iss.2, IET Software

[4] Idri.A and Abran.A,(2001): Towards A Fuzzy Logic Based Measures For Software Project similarity , In Proc. of the 7th International Symposium on Software Metrics, England, pp.85-96.

[5] Aron.J.D, (1969): Estimating Resource for Large Programming Systems, NATO Science Committee, Rome, Italy.

[6] Albrecht.A.J, and J. E. Gaffney.J.E, (1983): Software function, source lines of codes, and development effort prediction: a software science validation, IEEE Trans Software Eng. SE-9, pp.639-648.

[7] Zia.Z, Rashid.A, .Zaman.K.uz, (2009): Software cost estimation for component-based fourth generation-language, Vol.5, Iss.1, pp.103-110, IET Software.

[8] Ayman A. Issa, (2011): An Algorithmic Software Cost Estimation Model for Early Stages Of Software Development, International Journal of Academic Research, Vol. 3. No. 2.

[9] Kamaljit Kaur, Kirti Minhas, Neha Mehan, Namita Kakkar,(2009): Static and Dynamic Complexity Analysis of Software Metrics , World Academy of Science, Engineering and Technology

[10] Xishi Huang, Danny Ho, Jing Ren and Luiz F. Capretz, (2007): Improving the COCOMO model using a neuro-fuzzy approach, Applied Soft Computing, Vol. 7, pp.29–40.

[11] Joon-Kil Lee, Ki-Tae Kwon,(2009): Software Cost Estimation using SVR based on Immune Algorithm , 10th ACIS International conference on software engineering.

[12] Tron Foss et al., (2002): A simulation study of the model Evaluation Criterion MMRE

[13] Dan Port et al,(2009): Studies of Confidence in Software Cost Estimation Research Based on the Criterions MMRE and PRED , menzies.us/pdf/09predmmre.pdf.

[14] Parvinder S. Sandhu, Manisha Prashar, Pourush Bassi, and Atul Bisht,(2009): A Model for Estimation of Efforts in Development of Software Systems , World Academy of Science, Engineering and Technology, No.56.

[15] Pichai Jodpimai, Peraphon Sophatsathit and Chidchanok Lursinsap,(2010): Estimating Software Effort with Minimum Features using Neural Functional Approximation, ICCSA.

[16] Prasad Reddy P.V.G.D, Sudha K.R., Rama Sree P & Ramesh S.N.S.V.S.C, (May 2010):Software Effort Estimation using Radial Basis and Generalized Regression Neural Networks, Journal of Computing, Vol 2, Issue 5.

[17] Prasad Reddy P.V.G.D, Sudha K.R., Rama Sree P & Ramesh S.N.S.V.S.C,(June 2010): Fuzzy Based Approach for Predicting Software Development Effort, International Journal of Software Engineering, Vol .1, Issue 1.

[18] Harish Mittal (2007):Optimisation Criteria for effort estimation using fuzzy technique, CLEI Electronic journal, Vol 10, No 1.

[19] Mohammad Azzeh, Daniel Neagu, Peter I.Cowling,(2011): Analogy based software effort estimation using fuzzy numbers, The journal of systems and software ,Vol.84, 270-284.

[20] Jacky Keung, (2009): Software Development Cost Estimation using Analogy: A Review, Australian Software Engineering Conference.

[21] Yan-Fu Li, Min Xie, Thong-Ngee Goh,(2010): Adaptive ridge regression system for software cost estimating on multi-collinear datasets , The Journal of Systems and Software, Vol 83, pg. 2332-2343.

[22] Martin Auer, Stefan Biffl, (2004): Increasing the accuracy and reliability of Analogy-Based Cost Estimation with Extensive Project Feature Dimension Weighting, Proceedings of the ISESE.