International Journal of Computer Science and Engineering (IJCSE)
ISSN 2278-9960
Vol. 2, Issue 3, July 2013, 53-68

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

RELATIVE ANALYSIS OF SOFTWARE COST AND EFFORT ESTIMATION TECHNIQUES

BHAWANA SRIVASTAVA[1] & MANOJ WADHWA[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Echelon Institute of Technology, Faridabad, India

[2]Professor & HOD, Department of Computer Science and Engineering, Echelon Institute of Technology, Faridabad, India

ABSTRACT

Software effort estimation is a very critical task in the software engineering and to control quality and efficiency a suitable estimation technique is crucial. This paper gives a comparative analysis of various available software effort estimation techniques. These techniques can be widely categorised under algorithmic model, non-algorithmic model, parametric model, and machine learning models. The use of a model that accurately calculates the cost and effort of developing a software product can be a key to the success of whole development project. This paper presents a detailed analysis of several existing methods for software cost estimation. No single technique is best for all situations, and thus a careful comparison of the results of several approaches is most likely to produce realistic estimate.

KEYWORDS: Software Cost Estimation, Delphi, Software Effort Estimation, COCOMO, Parametric Model, Machine Learning

INTRODUCTION

Software effort estimation is one of the most critical and complex, but a key activity in the software development processes. Over the last three decades, a growing trend has been observed in using variety of software effort estimation models in diversified software development processes. It is realized that the importance of all these models lies in estimating the software development costs and preparing the schedules more quickly and easily in the anticipated environments. A great amount of research time and money have been devoted to improving accuracy of the various estimation models. There is no proof on software cost estimation models to perform consistently accurate within 25% of the actual cost and 75% of the time. The accuracy of the individual models decides their applicability in the projected environments, whereas the accuracy can be defined based on understanding the calibration of the software data.

Since the precision and reliability of the effort estimation is very important for the competitiveness of software companies, the enterprises and researchers have put their maximum effort to develop the accurate models to estimate effort near to accurate levels. Many estimation models have been proposed and can be categorized based on their basic formulation schemes; estimation by Non-algorithm methods expert [6], analogy based estimation schemes [6], algorithmic methods SLOC, FPA, COCOMO, SEER, SLIM including Machine Learning models like artificial neural network based approaches and fuzzy logic based estimation schemes. Accurate effort and cost estimation of software applications continue to be a critical issue for software project managers. Hence there are no best estimation methods for all different environments; they depend upon specific environment available.

ESTIMATION TECHNIQUES

Generally, there are many methods for software cost estimation, which are divided into four categories: Algorithmic, Non-Algorithmic, Parametric and Machine learning models. All categories is required for performing the

accurate estimation. If the requirements are known better, their performance will be better. In this section, some popular estimation methods are discussed.

Algorithmic Models

These models usually need data at first and make results by using the mathematical relations. Nowadays, many software estimation methods use these models. Algorithmic Models are classified into some different models like:

Source Line of Code: SLOC is an estimation parameter that illustrates the number of all commands and data definition but it does not include instructions such as comments, blanks, and continuation lines. After computing the SLOC for software, its amount is compared with other projects which their SLOC has been computed before, and the size of project is estimated. Thousand Lines of Code (KSLOC) are used for estimation in large scale. *SLOC Measuring seems very difficult at the early stages of the project because of the lack of information about requirements.*

Since SLOC is computed based on language instructions, comparing the size of software which uses different languages is too hard. Anyway, SLOC is the base of the estimation models in many complicated software estimation methods. SLOC usually is computed by

$$\mathbf{S = (S_{OPT}+4S_M+S_{PESS})/6}$$

Where , S = Estimated size, S_{OPT} = Optimistic Value, S_M = Most likely Value, S_{PESS} = Pessimistic Value

Function Point Analysis: Measuring software size in terms of line of code is analogous to measuring a car stereo by the number of registers, capacitors and integrated circuits involved in its production. At first, Alan Albrecht while working for IBM, recognized the problem in size measurement, and developed the technique which is called Function Point Analysis, which appeared to be a solution to the size measurement problem to measure the functionality of project. In this method, estimation is done by determination of below indicators:

- **User Inputs:** information entering the system.
- **User Outputs:** information leaving the system.
- **Logic Files:** information held within the system.
- **Enquiries:** requests for instant access to information.
- **Interfaces:** information held by other systems that are used by the system being analyzed.

Table 1: Functional Units and Weighting Factors

Functional Units	Weighting Factors		
	Simple	Medium	Complex
User Inputs	3	4	6
User Outputs	4	5	7
Logic files	3	4	6
Enquiries	7	10	15
Interfaces	5	7	10

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:

$$\mathbf{UFC= \sum_{i=1}^{5} \sum_{j=1}^{3} NijWij}$$

where Nij is the number of indicator i with complexity j and; Wij 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) which is obtained from the components in Table 1.

Table 2: Technical Complexity Factor Components

F1	Reliable back-up and recovery	F8	Data communications
F2	Distributed functions	F9	Performance
F3	Heavily used configuration	F10	Online data entry
F4	Operational ease	F11	Online update
F5	Complex interface	F12	Complex processing
F6	Reusability	F13	Installation ease
F7	Multiple sites	F14	Facilitate change

Each component can change from 0 to 5 and 0 indicate that the component has no effect on the project and the component is compulsory and very important respectively. Finally, the TCF is calculated as:

$$\mathbf{TCF = 0.65+0.01(\sum(Fi))}$$

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

$$\mathbf{FP=UFC*TCF}$$

Seer-Sem (Software Evaluation and Estimation of Resources - Software Estimating Model): SEER-SEM model has been proposed in 1980 by Galorath Inc (Galorath, 2006). Most parameters in this method are commercial and, business projects usually use SEER-SEM as their main estimation method. Size of the software is the most important feature in this method and a parameter namely Se is defined as effective size.

SEER-SEM has two main limitations on effort estimation:

First, there are over 50 input parameters related to the various factors of a project, which increases the complexity of SEER-SEM, especially for managing the uncertainty from these outputs.

Second, the specific details of SEER-SEM increase the difficulty of discovering the nonlinear relationship between the parameter inputs and the corresponding outputs. Overall, these two major limitations can lead to a lower accuracy in effort estimation by SEER-SEM.

Se is computed by determining the five indicators : newsize, existingsize, reimpl and retest as below:

$$\mathbf{Se=Newsize+ExistingSize(0.4Redesign+0.25r\ \ eimp+0.35Retest)}$$

After computing the Se the estimated effort is calculated as below:

$$\mathbf{Effort=D}^{0.4} * \left(\frac{Se}{Cte}\right)^{1.2}$$

Where,

- S_e is effective size - introduced earlier
- C_{te} is effective technology - a composite metric that captures factors relating to the efficiency or productivity with which development can be carried out. An extensive set of people, process, and product parameters feed into the effective technology rating. A higher rating means that development will be more productive

- D is staffing complexity - a rating of the project's inherent difficulty in terms of the rate at which staff are added to a project.

Once effort is obtained, duration is solved using the following equation:

$$T_d = D^{-0.2} * (Se/Cte)^{.4}$$

This equation relates the effective size of the system and the technology being applied by the developer to the implementation of the system. The technology factor is used to calibrate the model to a particular environment. This factor considers two aspects of the production technology – technical and environmental.

Cost Estimation Model: Early cost model were linear but it has been shown there is no clear linear relationship between effort and size.

Later cost models were generally based on the following non-linear formula:

$$E = (a + b*(SIZE^c)) * f(x_1 \ldots, x_n)$$

Base formula correction (depends on the value of entities ($x_1, \ldots x_n$)

Where,

E = effort a, b and c are derived constants and x_1 **to** x_n are influencing factors which vary from project to project.

There are three forms of the COnstructive COst MOdel :

- Basic CoCoMo which gives an initial rough estimate of man months and development time,
- Intermediate CoCoMo which gives a more detailed estimate for small to medium sized projects,
- Detailed CoCoMo which gives a more detailed estimate for large projects.

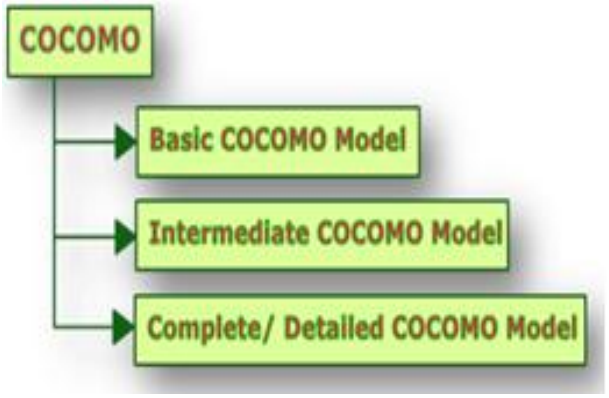

Figure 1: Types of COCOMO Model

DEVELOPMENT MODES

There are three modes of development:

- **Organic Mode**
  - Relatively Small, Simple Software projects.
  - Small teams with good application experience work to a set of less than rigid requirements.
  - Similar to previously developed projects.

- Relatively small and require little innovation.

- **Semidetached Mode**
  - Intermediate (in size and complexity) software projects in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements.

- **Embedded Mode**
  - Software projects that must be developed within set of tight hardware, software and operational Constraints.

- **Basic COCOMO:** Basic COCOMO (Constructive Cost Model) is an empirical estimation scheme proposed in 1981 [29] as a model for estimating effort, cost, and schedule for software projects. It was derived from the large data sets from 63 software projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/I. These data were analyzed to discover a set of formulae that were the best fit to the observations. These formulae link the size of the system and Effort Multipliers (EM) to find the effort to develop a software system. In COCOMO 81, effort is expressed as Person Months (PM) and it can be calculated as

  PM= a* Size^{b}*$\prod_{i=1}^{15} EMi$

  where,

"a" and "b" are the domain constants in the model. It contains 15 effort multipliers. This estimation scheme accounts the experience and data of the past projects, which is extremely complex to understand and apply the same. Cost drives have a rating level that expresses the impact of the driver on development effort, PM. These rating can range from Extra Low to Extra High. For the purpose of quantitative analysis, each rating level of each cost driver has a weight associated with it. The weight is called Effort Multiplier. The average EM assigned to a cost driver is 1.0 and the rating level associated with that weight is called Nominal.

- **COCOMO II:** In 1997, an enhanced scheme for estimating the effort for software development activities, which is called as COCOMO II. In COCOMO II, the effort requirement can be calculated as:

  PM= a* Size^{b}*$\prod_{i=1}^{17} EMi$

  Where,

  E=B+0.01*$\sum_{j=1}^{5}$ SFj

COCOMO II is associated with 31 factors; LOC measure as the estimation variable, 17 cost drives, 5 scale factors, 3 adaptation percentage of modification, 3 adaptation cost drives and requirements & volatility. Cost drives are used to capture characteristics of the software development that affect the effort to complete the project. COCOMO II used 31 parameters to predict effort and time [11] [12] and this larger number of parameters resulted in having strong co-linearity and highly variable prediction accuracy. Besides these meritorious claims, COCOMO II estimation schemes are having some disadvantages. The underlying concepts and ideas are not publicly defined and the model has been provided as a black box to the users [26]. This model uses LOC (Lines of Code) as one of the estimation variables, whereas Fenton et. al [27] explored the shortfalls of the LOC measure as an estimation variable. The COCOMO also uses FP (Function Point) as one of the estimation variables, which is highly dependent on development the uncertainty at the input level of the

COCOMO yields uncertainty at the output, which leads to gross estimation error in the effort estimation [33]. Irrespective of these drawbacks, COCOMO II models are still influencing in the effort estimation activities due to their better accuracy compared to other estimation schemes.

Table 3: Effort Multipliers

Attribute	Type	Description
RELY	Product	Required system reliability
CPLX	Product	Complexity of system modules
DOCU	Product	Extent of documentation required
DATA	Product	Size of database used
RUSE	Product	Required percentage of reusable components
TIME	Computer	Execution time constraint
PVOL	Computer	Volatility of development platform
STOR	Computer	Memory constraints
ACAP	Personnel	Capability of project analysts
PCON	Personnel	Personnel continuity
PCAP	Personnel	Programmer capability
PEXP	Personnel	Programmer experience in project domain
AEXP	Personnel	Analyst experience in project domain
LTEX	Personnel	Language and tool experience
TOOL	Project	Use of software tools
SCED	Project	Development schedule compression
SITE	Project	Extent of multisite working and quality of inter-site communications

Table 4: Scale Factors

Factor	Explanation
Precedentedness (PREC)	Reflects the previous experience of the organization
Development Flexibility (FLEX)	Reflects the degree of flexibility in the development process.
Risk Resolution (RESL)	Reflects the extent of risk analysis carried out.
Team Cohesion (TEAM)	Reflects how well the development team knows each other and work together.
Process maturity (PMAT)	Reflects the process maturity of the organ

- **The Detailed COCOMO Model:** The detailed model differs from the Intermediate model in only one major aspect: the detailed model uses different Effort Multipliers for each phase of a project. These phase dependent Effort Multipliers yield better estimates than the Intermediate model. The six phases COCOMO defines are:

Table 5: Phases Table

Abbreviation	Phase
RQ	Requirements
PD	Product Design
DD	Detailed Design
CT	Code & Unit Test
IT	Integrate & Test
MN	Maintenance

The phases from Product Design through Integrate & Test are called the Development phases. Estimates for the Requirements phase and for the Maintenance phase are performed in a different way than estimates for the four Development phases. The Programmer Capability cost driver is a good example of a phase dependent cost driver. The Very High rating for the Programmer Capability Cost Driver corresponds to an Effort Multiplier of 1.00 (no influence) for the Product Design phase of a project, but an Effort Multiplier of 0.65 is used for the Detailed Design phase. These ratings indicate that good programmers can save time and money on the later phases of the project, but they don't have an impact on the Product Design phase because they aren't involved.

Example: A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- Database part
- Graphical User Interface (GUI) part
- Communication part

Of these, the communication part can be considered as **Embedded software**. The database part could be **Semi-detached software**, and the GUI part **Organic software**. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

SLIM Estimation Model: SLIM Software Life-Cycle Model was developed by Larry Putnam [28]. SLIM hires the probabilistic principle called Rayleigh distribution between personnel level and time. SLIM is basically applicable for large projects exceeding 70,000 lines of code.

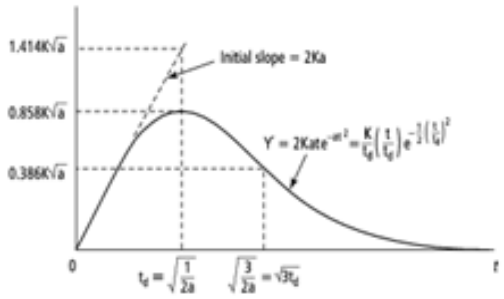

Figure 2: The Rayleigh Curve for SLIM

It makes use of Rayleigh curve referred from [14] as shown in figure 1 for effort prediction. This curve represents manpower measured in person per time as a function of time. It is usually expressed in personyear/ year (PY/YR). It can be expressed as:

$$\frac{dy}{dt} = 2\,Kate^{-2at2}$$

dy/dt is the manpower utilization per unit time, " t" is the elapsed time, "a" is the parameter that affects the shape of the curve and "K" is the area under the curve. There are two important terms associated with this curve:

- Manpower Build up given by D0=K/td3
- Productivity = Lines of Code/ Cumulative Manpower i.e.P=S/E and S= CK1/3 td 4/3,where C is the technology factor which reflects the effects of various factors on productivity such as hardware constraints, program complexity, programming environment and personal experience.

The SLIM Model Uses Two Equations: the software the manpower equation and software productivity level equation The SLIM model uses Rayleigh distribution to estimate to estimate project schedule and defect rate. Two key attributes used in SLIM method are productivity Index (PI) and Manpower Build up Index (MBI). The PI is measure of process efficiency (cost-effectiveness of assets), and the MBI determines the effects on total project effort that result from variations in the development schedule [A Probabilistic Model].

Inputs Required: To use the SLIM method, it is necessary to estimate system size, to determine the technology factor, and appropriate values of the manpower acceleration. Technology factor and manpower acceleration can be calculated using similar past projects. System size in terms of KDSI is to be subjectively estimated. This is a disadvantage,

because of the difficulty of estimating KDSI at the beginning of a project and the dependence of the measure on the programming language.

Completeness of Estimate: The SLIM model provides estimates for effort, duration, and staffing information for the total life cycle and the development part of the life cycle. COCOMO I provides equations to estimate effort, duration, and handles the effect of re-using code from previously developed software. COCOMO II provides cost, effort, and schedule estimation, depending on the model used (i.e., depending on the degree of product understanding and marketplace of the project). It handles the effect of reuse, reengineering, and maintenance adjusting the used size measures using parameters such as percentage of code modification, or percentage of design modification.

Assumptions: SLIM assumes the Rayleigh curve distribution of staff loading. The underlying Rayleigh curve assumption does not hold for small and medium sized projects. Cost estimation is only expected to take place at the start of the design and coding, because requirement and specification engineering is not included in the model.

Complexity: The SLIM model's complexity is relatively low. For COCOMO the complexity increases with the level of detail of the model. For COCOMO I the increasing levels of detail and complexity are the three model types: basic, intermediate, and detailed. For COCOMO II the level of complexity increases according to the following order: Application Composition, Early Design, Post Architecture.

Automation of Model Development: The Putnam method is supported by a tool called SLIM (Software Life-Cycle Management). The tool incorporates an estimation of the required parameter technology factor from the description of the project. SLIM determines the minimum time to develop a given software system. Several commercial tools exist to use COCOMO models.

Application Coverage: SLIM aims at investigating relationships among staffing levels, schedule, and effort. The SLIM tool provides facilities to investigate trade-offs among cost drivers and the effects of uncertainty in the size estimate.

Generalizability: The SLIM model is claimed to be generally valid for large systems. COCOMO I was developed within a traditional development process, and was a priori not suitable for incremental development. Different development modes are distinguished (organic, semidetached, embedded). COCOMO II is adapted to feed the needs of new development practices such as development processes tailored to COTS, or reusable software availability. No empirical results are currently available regarding the investigation these capabilities.

Comprehensiveness: Putnam's method does not consider phase or activity work breakdown. The SLIM tool provides information in terms of the effort per major activity per month throughout development. In addition, the tool provides error estimates and feasibility analyses. As the model does not consider the requirement phase, estimation before design or coding is not possible. Both COCOMO I and II are extremely comprehensive. They provide detailed activity distributions of effort and schedule. They also include estimates for maintenance effort, and an adjustment for code re-use. COCOMO II provides prototyping effort when using the Application Composition model. The Architectural Design model involves estimation of the actual development and maintenance phase. The granularity is about the same as for COCOMO I.

Non Algorithmic Methods

Contrary to the Algorithmic methods, methods of this group are based on analytical comparisons and inferences. For using the Non Algorithmic methods some information about the previous projects which are similar the under estimate project is required and usually estimation process in these methods is done according to the analysis of the previous

datasets. Here, three methods have been selected for the assessing because these methods are more popular than the other None Algorithmic methods and many papers about their usage have been published in the recent years (Idri, Mbarki et al. 2004; Braz and Vergilio 2006; Li, Xie et al. 2007; Keung, Kitchenham et al. 2008; Li, Lin et al. 2008; Jianfeng, Shixian et al. 2009; Jorgensen, Boehm et al. 2009.

- **Analogy:** It means creating estimates for new projects by comparing the new projects to similar projects from the past. As the algorithmic techniques have a disadvantage of the need to calibrate the model. So, the alternative approach is "analogy by estimation". But it requires considerable amount of computation. This process is much simple. But not all organizations have historical data to satisfactorily use analogy as means of estimation. ISBSG (International Software benchmarking Standards Group) maintains and exploits a repository of International Software Project Metrics to help software and IT business customers with project estimation; risk analysis, productivity, and benchmarking [25].
- **Expert Judgment:** Estimation based on Expert judgment is done by getting advices from experts who have extensive experiences in similar projects. This method is usually used when there is limitation in finding data and gathering requirements. Consultation is the basic issue in this method. One of the most common methods which work according to this technique is Delphi. Delphi arranges an especial meeting among the project experts and tries to achieve the true information about the project from their debates.[25] Delphi includes some steps:
    - The coordinator gives an estimation form to each expert.
    - Each expert presents his own estimation (without discussing with others)
    - The coordinator gathers all forms and sums up them (including mean or median) on a form and ask experts to start another iteration.
    - Steps (ii-iii) are repeated until an approval is gained.

Figure shows an example of using Delphi technique in which eight experts contributed and final convergence was determined after passing four stages.

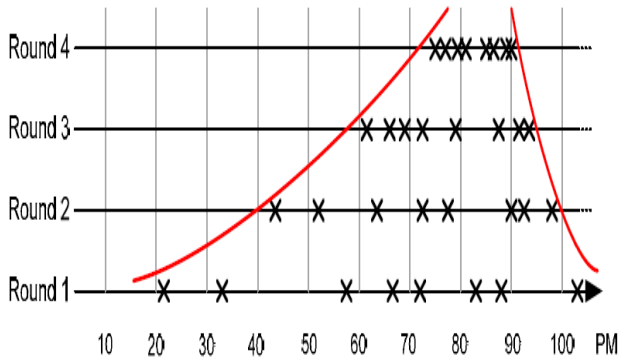

Figure 3: An example of Using Delphi

Parametric Models

Use effort drivers representing characteristics of the target system and the implementation environment used to predict the new effort. In the top-down approach, model is used to produce overall estimation using effort driver. And bottom-up approach is no past project data is available, and then we use the parametric model. Here break project into smaller and smaller components. Estimate costs for the lowest level activities and using lowest level calculate the higher level estimation. This model based on historical data about the software project. It will find the time factors affecting the

project to complete the time and effort estimation. The parameters are the personnel, programmer skill set, tools to develop a software and reuse factors.

An estimation method is classified as 'Framework based' when the following 2 characteristics are satisfied.

- A defined technique is integrated within the method.
- A history of similar projects is integrated within the method.

Brake Down Estimation [30]

In this method, the total project work is divided into several minute components and estimation is done at the level of components. There can be 3 different types of estimation methods possible depending on how we map the productivity measures into the estimation process.

- **Complex Productivity Estimation:** The following steps are adopted to arrive at an estimate in the case of 'Complex Productivity' estimation method.
    - The System or the project work is divided into 'n' no. of components.
    - For each component, no. of test cases required to test is estimated. The total no. of test cases required is divided further into 3 levels of complexities HIGH, MEDIUM and LOW. There will be standard definitions available to classify a test case into these 3 categories at the project, domain or technology level.
    - There is a productivity table available giving 'Effort / Test case' ratio for each Component-Complexity combination. This table is derived out of historical data and will have an effort value for each of the complexities of test case against each component in the system.
    - The testing effort of a component is the sum-product of effort values of different complexity levels and the no. of test cases in each complexity level.
    - The testing effort of the total system is the sum of testing efforts of all components.

This method is applicable when the components in a system are of widely varied nature which necessitates complexity definitions and analysis at the component level rather than at the system level.

- **Simplex Productivity Estimation***:* The following steps are adopted to arrive at an estimate in the case of 'Simplex Productivity' estimation method. The first two steps are similar to that of Complex Productivity estimation.
    - There is a productivity table giving 'Effort / Test case' ratio for each of the Test case complexity level. This table is derived out of historical data and will have an effort value for each of the complexity levels of test case wrt the over-all system.
    - The testing effort of a component is the sum-product of effort values of different complexity levels and the no. of test cases in each complexity level.
    - The testing effort of the system is the sum of testing efforts of all components.

This method is applicable where we have LOW test case equivalent formulae available for medium and high test case complexities.

- **Simple Productivity Estimation***:* The following steps are adopted to arrive at an estimate in the case of 'Simple Productivity' estimation method. The first two steps are similar to Complex Productivity estimation.
    - There is a productivity table giving 'Effort / Test case' ratio for LOW complexity Test cases. This table is derived out of historical data. The table will have an effort value for low complexity test case wrt the over-all system.
    - The estimate for a component is arrived as follows: Determine the LOW, MEDIUM and HIGH complexity test cases for the component. From this the equivalent LOW complexity test case count is calculated. The testing effort for the component is the product of 'Effort / Test Case' ratio as in step 3 and the equivalent LOW test case count.
    - The testing effort of the system is the sum of testing efforts of all components.

This method is applicable where we have LOW test case equivalent formulae available for medium and high test case complexities. By tweaking the method a little bit, the estimator can be given the freedom for putting a range of LOW equivalent TCs for say Medium complexity to get a very close approximation to the reality in the case of a particular component. For example, even if the historic table mentions that the MEDIUM / LOW ratio is 1.5, the estimator can put a value of 1.45 for a particular component to have a close approximation in the case of a project in hand.

Machine Learning Model

During the last two decades researchers have been focused on exploring a new approach using AI based techniques for accurate effort estimation. This approach uses ML a sub field of AI.

It is difficult to determine which technique gives more accurate result on which dataset. However, a lot of research has been done in Machine learning techniques of estimation and Literature suggests that ML methods are capable of providing adequate estimation models as compared to the traditional models especially in GSD projects [11]-[22].

ML algorithms offer a practical alternative to the existing approaches to many SE issues.

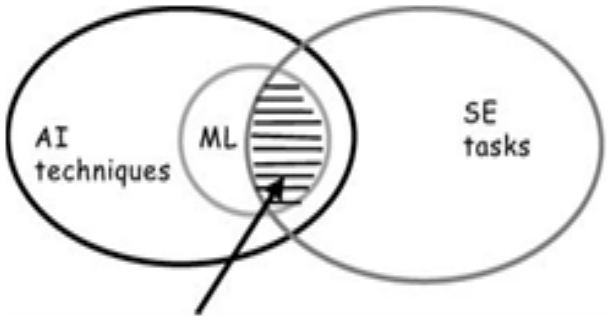

Figure 4: Relation between ML and Software Engineering

During last two decades Artificial Intelligence based models are attracting researcher's attention for the estimation of software parameters. In 1995 [23] have compared AI based techniques with traditional COCOMO, Function Point Analysis and Software Lifecycle Management (SLIM) and concluded that AI models are viable to traditional methods. Authors of the paper [24] have concluded that AI based models are capable of providing acceptable estimation models.

Below we will describe some commonly used methods of ML for measuring effort and in the next section we will compare these methods, so that this paper may help the practitioners and researchers in the selection of suitable effort estimation methods. Commonly used ML methods for measuring effort in GSD projects are as follows:

Artificial Neural Network (ANN): ANN is a computational or mathematical model that is stimulated by the biological human brain. Through learning process ANN can be configured for a specific application, such as pattern recognition or data classification. ANNs include the two basic components of biological neural networks that are Neurons (nodes) and Synapses (weights). A neuron has a set of *n* (number neurons in previous layer) synapses (inputs), which are characterized by *n* different weight (free parameters).[24]

Feed-Forward Neural Network (FFNN)

Many neurons are used in the construction of an FFNN; these neurons are connected with each other through specific network architecture. The primary goal of the FFNN is to transform the inputs into meaningful outputs. There is no self loop or backward feed in this network [24].

Back propagation neural network is the best selection for software estimation problem because it adjusts the weights by comparing the network outputs and actual results. In addition, training is done effectively. Majority of researches on using the neural networks for software cost estimation, are focused on modeling the Cocomo method, for example in (Attarzadeh, Ow, 2010) a neural network has been proposed for estimation of software cost according to the following figure. Scale Factors (SF) and effort multipliers (EM) are input of the neural network, p_i and q_j are respectively the weight of SFs and EMs.[32]

Fuzzy Method: All systems, which work based on the fuzzy logic try to simulate human behavior and reasoning. In many problems, which decision making is very difficult and conditions are vague, fuzzy systems are an efficient tool in such situations. This technique always supports the facts that may be ignored. There are four stages in the fuzzy approach:

Stage 1: Fuzzification: to produce trapezoidal numbers for the linguistic terms.

Stage 2: To develop the complexity matrix by producing a new linguistic term.

Stage 3: To determine the productivity rate and the attempt for the new linguistic terms.

Stage 4: Defuzzification: to determine the effort required to complete a task and to compare the existing method.

COMPARISON OF THE ESTIMATION METHODS

This section compares the mentioned estimation methods based on these advantages and disadvantages. This comparison con be useful for choosing an appropriate method in a particular project in a particular environment. Selection of the estimation technique is based on capabilities of methods and state of the project.

This table explains that many models are present but all are dependent on different environments and needs of the companies. But in all methods maximum LOC and FPA play the main or basic role for estimation.

We cannot always estimate by these so different kinds of factors included in estimation techniques are based on statistics, predictions like regression, expert's contribution, historical data sets, and Neural Network and Fuzzy logics. Seer-SEM and SLIM model are easy to implement by machine learning methods but lots of calculation and training is required which is not feasible for all situations.

These models are used in large organisations like for manufacturing, hardware, electronics and systems, trading and so on. The use of models not only depends upon factors of the methods but also upon the companies or organisation.

Table 6 shows a comparison of mentioned methods for estimation. For making a comparison, the popular existing estimation methods have been selected.

Table 6: Comparison of the Existing Methods

Sr.No	Method	Type	Advantages	Disadvantages
1	LOC	Algorithmic	Very easy in implementation to estimate the size of software	Prediction of line is tough in early stage, not good for very large project and Language dependent
2	Functional point	Algorithmic	Applied early in SDLC.GUI based, better than LOC, language free	Lots of judgement involved, start after the design specification, Less research data is available on function
3	SEER-SEM	Algorithmic	Used in very large projects	50 input parameters are required which increased the complexity and uncertainty
4	Basic COCOMO	Algorithmic	*Basic COCOMO* is good for quick, early, rough order of magnitude estimates of software costs, commonly used in small projects, compatible for assemble language to PL/I.	Not used in large projects where size is greater than 10000. Accuracy is limited. Its prediction is .25 which is quite poor
5	COCOMO II	Algorithmic	It provides more support for modern software development processes and an updated project database. Provide support to mainframe, code reusability and batch processing.	It cannot estimate the effort at all the different phases of SDLC. Its prediction is .68 which is quite good.
6	Detailed COCOMO	Algorithmic	**Phase Sensitive** effort multipliers are each to determine the amount of effort required to complete each phase.	Lots of parameters involved in estimation time complexity is high. Its prediction is .70 which is good.
7	Linear model	Algorithmic	It is a best method of prediction using linear regression technique	Little difference between actual and predicted result and error is also need to calculate.
8	SLIM	Algorithmic	A Probabilistic Model, Used in a very large project	For only large projects
9	Expert Judgment	Non-Algorithmic	Fast prediction, Adapt to especial projects	Its success depend on expert, Usually is done incomplete
10	Analogy	Non-Algorithmic	Works based on actual experiences, having especial expert is not important	A lots of information about past projects is required, In some situations there are no similar project
11	Complex productivity model	Parametric	It is useful when components in a system are of widely varied in nature	In it analysis is done at component level rather than at the system level.
12	Simplex productivity model	Parametric	This method is applicable when the components in a system are of similar nature	Extra intervention of experts is required to determine the effort values of High, Medium and Low complexity test cases at the system level.
13	Simple productivity model	Parametric	This method is applicable where we have LOW test case equivalent formulae available for medium and high test case complexities.	Expert or Estimator can predict the approximate values by the help of historical dataset.
14	Neural Networks	Machine learning model	Consistent with unlike databases, Power of reasoning	There is no guideline for designing, The performance depends on large training data
15	Fuzzy	Machine learning model	Training is not required, Flexibility	Hard to use, Maintaining the degree of meaningfulness is difficult

CONCLUSIONS

Based upon the background readings and some pronounced case studies of companies, it is found that the existing models are highly credible; however, this survey states that this is not so. All the models cannot predict the actual either against the calibration data or validation data to any level of accuracy or consistency. Surprisingly, SEER and machine

learning techniques are reliable and good at predicting the effort. These days, all the leading organisations are using their proprietary effort estimation software or automated software tools for conducting estimations and thus do not require complete estimation techniques for the estimation. Now companies conduct market survey and understand customer's requirement and according to that they state their requirements to developers and then developers analyze the customer requirement and customise the existing software, update the software requirement, append some new modules into the software and so on. For this, they use some kind of mix of estimation techniques which are based on some of these models.

This varies from company to company as to what project they mostly deals with and according to that they create their own software for the estimation through which they get correct estimation for the projects however, the existing models are not so accurate because they lie in the term prediction; prediction never comes true is proved in this estimation models. In all the models, the two key factors that influence the estimate are project size either in terms of LOC or FP and the capabilities of the development team personnel. Finally, this paper concludes that many good techniques and methods exist which can suffice in different situations but none is best and suitable for every type of situation or requirement. Thus there is a need for an adequate mix and use of these techniques according to the changing requirement so as to provide the best estimation.

REFERENCES

1. Albrecht.A.J. and J. E. Gaffney, *"Software function, source lines of codes, and development effort prediction: a software science validation",* IEEE Trans Software Eng. SE,pp.639-648, 1983
2. Ali Idri, Alain Abran, Taghi M. Khosgoftaar. 2001. *"Fuzzy Analogy- A New Approach for Software Cost Estimation".* International Workshop on Software Measurement (IWSM'01).
3. Allan J. Alberecht and John E. Gaffhey, November 1983*, "Software Function, Source Lines of Code and Development Effort Prediction : A software Science Validation"* . IEEE transactions on Software Engineering.
4. Allan J. Alberecht, May 1984. *"AD/M Productivity Measurement and Estimation Validation, IBM Corporate Information Systems".* IBM Corp.
5. Attarzadeh,I. Siew Hock Ow, *"Proposing a New Software Cost Estimation Model Based on Artificial Neural Networks",*IEEE International Conference on Computer Engineering and Technology (ICCET) , Volume: 3, Page(s): V3-487 - V3-491 2010.
6. Attarzadeh, I. Siew Hock Ow, "*Improving the accuracy of software cost estimation model based on a new fuzzy logic model",*World Applied science journal 8(2):117-184,2010-10-2.
7. Banker, R. D., H. Chang, et al. (1994). *"Evidence on economies of scale in software development".* Information and Software Technology .
8. Bergeron, F. and J. Y. St-Arnaud (1992). *"Estimation of information systems development efforts: a pilot study".* Information and Management 22(4): 239-254.
9. Boehm, B. W. and P. N. Papaccio (1988). *" Understanding and controlling software costs".* IEEE Transactions on Software Engineering 14(10): 1462-1477.
10. Boehm, C Abts, and S Chulani. *"Software Development Cost Estimation Approaches – A Survey",* Technical Report USC-CSE- 2000-505", University of Southern California – Center for Software Engineering, USA, (2000).

11. B.W. Boehm, *"Software Engineering Economics,"* Prentice Hall, 1981.
12. B.W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *"Software Cost Estimation with COCOMO II,"* Prentice Hall, 2000.
13. Capers Jones, Chief Scientist Emeritus Software Productivity Research LLC. . How Software Estimation Tools Work. Version 5 – February 27, 2005Charles Symons 1991.
14. Capers Jones, *" Software Sizing and Estimation Mark II function Points (Function Point Analysis*)", Wiley 1991.
15. Chatzoglou, P. D. and L. A. Macaulay (1998). *"A rule-based approach to developing software development .estimation using computational intelligence techniques"*. Nature & Biologically Inspired Computing, NaBIC 2009. World Congress on,2009.
16. Chiu NH, Huang SJ, *"The Adjusted Analogy-Based Software Effort Estimation Based on Similarity Distances,"* Journal of Systems and Software, Volume 80, Issue 4, pp 628-640, 2007
17. Chintala Abhishek, Veginati Pavan Kumar, Harish Vitta, Praveen Ranjan Srivastava, *" Test Effort Estimation Using Neural Network",* J. Software Engineering & Applications, 2010, 3: 331-340
18. Heiat A, *"Comparison of Artificial Neural Network and Regression Models for Estimating Software Development Effort,"* Journal of Information and Software Technology, Volume 44, Issue 15, pp 911-922, 2002
19. Huang SJ, Lin CY, Chiu NH, *"Fuzzy Decision Tree Approach for Embedding Risk Assessment Information into Software Cost Estimation Model," Journal* of Information Science and Engineering, Volume 22, Number 2, pp 297–313, 2006
20. Huang Kaczmarek J, Kucharski M, *"Size and Effort Estimation for Applications Written in Java,"* Journal of Information and Software Technology, Volume 46, Issue 9, pp 589-60, 2004
21. Jovan Popovic and Dragan Bojic, "*A Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle",* Com SIS Vol. 9, No. 1, January 2012
22. Jorgen M, Sjoberg D.I.K, *"The Impact of Customer Expectation on Software Development Effort Estimates"* International Journal of Project Management, Elsevier, pp 317-325, 2004
23. Jeffery R, Ruhe M,Wieczorek I, *"Using Public Domain Metrics to Estimate Software Development Effort,"* In Proceedings of the 7th International Symposium on Software Metrics, IEEE Computer Society, Washington, DC, pp 16–27, 2001.
24. Mamoona Humayun and Cui Gang, *"Estimating Effort in Global Software Development Projects Using Machine Learning Techniques",* International Journal of Information and Education Technology, Vol. 2, No. 3, June 2012.
25. P.K. Suri, Pallavi Ranjan, "*Comparative Analysis of Software Effort Estimation Techniques*, International Journal of Computer Applications (0975 – 8887) Volume 48– No.21, June 2012.
26. Rathi.j,Kamalraj. R , Karthik. S, *"Survey on Effective Software Effort Estimation Techniques"* International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) ISSN: 2278 – 1323, Volume 1, Issue 8, October 2012
27. R. Jensen, *"An improved macrolevel software development resource estimation model".* In 5th ISPA Conference, pp 88–92, 1983.

28. Saleem Basha, Dhavachelvan P, "*Analysis of Empirical Software Effort Estimation Models*", (IJCSIS) International Journal of Computer Science and Information Security,Vol. 7, No. 3, 2010

29. Satyananda, "*An Improved Fuzzy Approach for COCOMO's Effort Estimation Using Gaussian Membership Function"* Journal of Software, vol 4, pp 452-459, 2009.

30. S. chulani, B. Boehm, and B. Steece*, "Bayesian Analysis of Emperical Software Engineering Cost Models*," IEEE Trans. Software Eng., vol.25, no. 4, pp.573-583, 1999.

31. Sikka, G., A. Kaur, et al. *"Estimating function points: Using machine learning and regression models".* Education Technology and Computer (ICETC), 2nd International Conference on,2010.

32. Vahid Khatibi, Dayang N. A. Jawawi, *" Software Cost Estimation Methods: A Review"*, Journal of Emerging Trends in Computing and Information Sciences , Volume 2 No. 1 ,ISSN 2079-8407.

33. Vu Nguyen, Bert Steece, Barry Boehm *"A Constrained Regression Technique for COCOMO Calibration"* ESEM'08, ACM, pp 213-222, 2008.

34. Wei Lin Du, Danny Ho, Luiz Fernando Capretz , " *Improving Software Effort Estimation Using Neuro-Fuzzy Model with SEER-SEM"*,Global Journal of Computer Science and Technology Vol. 10 Issue 12 (Ver. 1.0) October 2010.

35. Yinhuan, Z., W. Beizhan, et al. *"Estimation of software projects effort based on function point".* Computer Science & Education. ICCSE. 4th International Conference on,2009.

36. Ziauddin, Shahid Kamal Tipu, Shahrukh Zia, "*An Effort Estimation Model for Agile Software Development"* Advances in Computer Science and its Applications (ACSA) Vol. 2, No. 1, 2012, ISSN 2166-2924.