International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 12, December 2014 **ISSN 2319 - 4847**

Software Effort Estimation: A Comparison Based Perspective

Poonam Rijwani[1], Sonal Jain[2] and Dharmesh Santani[3]

[1] Computer Science Department, Pratap University, Jaipur, Rajasthan

[2] Computer Science Department, JK Lakshmipat University, Jaipur, Rajasthan

[3] Computer Engineering Department, Poornima University, Jaipur, Rajasthan

ABSTRACT

For any industry to stay competitive, managing balance between quality and cost of software is important. Estimating software development effort remains a complex problem and one which continues to attract considerable research attention. Number of researchers had made their efforts to produce different modeling techniques in last few decades. This paper is about the comprehensive descriptive exploration of the techniques that were presented in software effort estimation field. In this paper we present the main findings of few research papers that have utilized a various parametric and non parametric techniques amalgamated with computational intelligence technique, in software effort estimation. All widespread models discussed at one place will give researchers a prospect to comprehend the pros and cons, similarities and the differences among various models.

Keywords:- software effort estimation, algorithmic models, software Metrics; line of code, computational intelligence, neural networks, back propagation, mean magnitude of relative error.

1. INTRODUCTION

According to Roger S Pressman "Software efforts estimation is not an exact science". Day after day competition in the software industries is increasing; in such scenario accurate efforts estimation has become an important task. Effort estimation is one of the essential activities of software development which is providing basis for other software activities like scheduling and planning [1]. Software cost and efforts estimation is become a challenge for IT industries. There are lots of methods existing for efforts and cost estimation, but people do not know how to use these methods. IT industries are developing two types of projects. First, which has a specific customer; all requirements are given by the customer. Second, in which requirements are gathered by using some survey. In case of second type of projects, efforts estimation becomes much difficult because we do not have a specific customer and we have to estimate efforts accurately and complete the project within time limit. So, a systematic approach is required for effort estimation. Accurate cost estimation is important because it can help to classify and prioritize development projects to determine what resources to commit to the project and how well these resources will be used. An extremely helpful form of effort prediction is the one made at an early stage during a project [2]. However, estimates at the preliminary stages of the project are the most difficult to obtain because the primary source to estimate the costing comes from the requirement specification document. There are number of competing software cost estimation methods available for software developers to predict effort and test effort required for software development, from the intuitive expert opinion methods to the more complex algorithmic modeling methods and the analogy-based methods [2], [11], [17], [19]. Software effort estimation models can be categorized into two main categories: algorithmic and non-algorithmic. The algorithmic methods have been largely studied and there are a lot of models have been developed, such as COCOMO models [3], Putnam model [5], and function points based models [4]. The selected papers were classified according to applied evaluation strategies, and measurement perspectives.

Various effort estimation methods

Expert Judgment Method

Expert judgment techniques involve consulting with software cost estimation expert or a group of experts to use their experience and understanding of the proposed project to arrive at an estimate of its cost. For coordination of differing opinions among estimators, often used is one of formal techniques like Delphi. There exist a number of Delphi technique forms. Wideband Delphi encourages those involved to discuss the problem among them. This technique is implemented in following steps:

1. Coordinator will acquaint every expert with project specifications and estimation manner.
2. Coordinator will call a meeting of experts to discuss the issues related to the value to be estimated.
3. Each expert will individually and independently complete the form.
4. Coordinator will call another meeting to discuss mainly the estimates that differ most from the others.

5. Experts will complete the forms again.
6. Steps 4 and 5 will be repeated until consensus has been reached.

The advantages of this method are:

1. The experts can factor in differences between past project experience and requirements of the proposed project.
2. The experts can factor in project impacts caused by new technologies, architectures, applications and languages involved in the future project and can also factor in exceptional personnel characteristics and interactions, etc.

The disadvantages include:

1. This method cannot be quantified.
2. It is hard to document the factors used by the experts or experts-group.
3. Expert may be some biased, optimistic, and pessimistic, even though they have been decreased by the group consensus.
4. The expert judgment method always compliments the other cost estimating methods such as algorithmic method.

Estimating by Analogy

The underlying reasoning behind analogy is similar to a basic human reasoning process used by almost every individual on a daily basis to solve problems based upon similar event(s) that happened in the past. In software engineering, we use Analogy to utilize the specific knowledge of previously experienced, concrete problem situations or cases. A solution is derived by finding in similar case(s), and reusing it in a new problem situation.
The main advantages of this method are:

1. The estimation is based on actual project characteristic data. The estimator's past experience and knowledge can be used which is not easy to be quantified.
2. The differences between the completed and the proposed project can be identified and impacts estimated.

However there are also some problems with this method:

1. Using this method, we have to determine how best to describe projects. The choice of variables must be restricted to information that is available at the point that the prediction required.
2. Possibilities include the type of application domain, the number of inputs, the number of distinct entities referenced, the number of screens and so forth.

Top-Down Estimating Method

Top-down estimating method is also called Macro Model. Using top-down estimating method, an overall cost estimation for the project is derived from the global properties of the software project, and then the project is partitioned into various low-level components. The leading method using this approach is Putnam model. This method is more applicable to early cost estimation when only global properties are known. In the early phase of the software development, it is very useful because there is no detailed information available. The advantages of this method are:

1. It focuses on system-level activities such as integration, documentation, configuration management, etc., many of which may be ignored in other estimating methods and it will not miss the cost of system-level functions.

2. It requires minimal project detail, and it is usually faster, easier to implement.

The disadvantages are:

1. It often does not identify difficult low-level problems that are likely to escalate costs and sometime tends to overlook low-level components.

2. It provides no detailed basis for justifying decisions or estimates. Because it provides a global view of the software project, it usually embodies some effective features such as cost-time trade off capability that exists in Putnam model.

Bottom-up Estimating Method

Using bottom-up estimating method, the cost of each software components is estimated and then combines the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge accumulated about the small software components and their interactions. The leading method using this approach is COCOMO's detailed model.

The advantages are:

1) It permits the software group to handle an estimate in an almost traditional fashion and to handle estimate components for which the group has a feel.

2) It is more stable because the estimation errors in the various components have a chance to balance out.

The disadvantages are:

1. It may overlook many of the system-level costs (integration, configuration management, quality assurance, etc.) associated with software development.
2. It may be inaccurate because the necessary information may not available in the early phase.
3. It tends to be more time-consuming.
4. It may not be feasible when either time or personnel are limited.

Algorithmic Methods

The algorithmic method is designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and 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, skill-levels, risk assessments, etc. The algorithmic methods have been largely studied and there are a lot of models have been developed, such as COCOMO models [3], Putnam model [5], and function points based models [6].

General advantages:

1. It is able to generate repeatable estimations.
2. It is easy to modify input data, refine and customize formulas.
3. It is efficient and able to support a family of estimations or a sensitivity analysis.
4. It is objectively calibrated to previous experience.

General disadvantages:

1. It is unable to deal with exceptional conditions, such as exceptional personnel in any software cost estimating exercises, exceptional teamwork, and an exceptional match between skill-levels and tasks.
2. Poor sizing inputs and inaccurate cost driver rating will result in inaccurate estimation.
3. Some experience and factors cannot be easily quantified.

COCOMO

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.

Boehm proposed three levels of the model: basic, intermediate, detailed.

- The **basic COCOMO'81** model is a single-valued, static model that computes software development effort (and cost) as a function of program size expressed in estimated thousand delivered source instructions (KDSI).
- The **intermediate COCOMO'81** model computes software development effort as a function of program size and a set of fifteen "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.
- The **advanced or detailed COCOMO'81** 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.

COCOMO'81 models depend on the two main equations:

1. Development effort: $MM = a * KDSI^{b}$ (1)

Based on MM - man-month / person month / staff-month is one month of effort by one person. In COCOMO'81, there are 152 hours per Person month. According to organization this values may differ from the standard by 10% to 20%.

2. Effort and development time (TDEV): $TDEV = 2.5 * MM^{c}$ (2)

The coefficients a, b and c depend on the mode of the development. There are three modes of development:

Development Mode	Project Characteristics			
	Size	Innovation	Deadline/constraints	Dev. Environment
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/ customer interfaces

Fig 1: Project Characteristics of modes of development

Basic COCOMO

The basic COCOMO applies the parameterised equation without much detailed consideration of project characteristics.

$MM = a * KDSI^{b}$

$TDEV = 2.5 * MM^{c}$

Basic COCOMO	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Fig 2: Effort equations and constants for Basic model

Intermediate COCOMO

The same basic equation for the model is used, but fifteen cost drivers are rated on a scale of 'very low' to 'very high' to calculate the specific 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 for a cost driver that's judged as normal.

Software Product

- Reliability requirement
- Database Size
- Product Complexity

Computer System

- Execution Time Constraint
- Main Storage Constraints
- Virtual Machine volatility
- Computer Turnaround Time

Human Resource

- Analyst capability
- Virtual Machine Experience
- Programmer capability
- Programming Language Experience
- Application Experience

Software Project

- Use of modern programming practices
- Use of Software Tools
- Required Development Schedule

Fig 3: Factors of Intermediate COCOMO.

In the intermediate COCOMO model, the effort estimate value generated by the basic COCOMO model is multiplied with the corresponding effort multiplier to refine the estimation:

$\mathbf{Effort_f = Effort_i \times EM_f}$

where;

$\mathbf{Effort_f}$: Refined effort value for the factor f (expressed in PMs).

$\mathbf{Effort_i}$: Initial effort value calculated with the basic COCOMO model (expressed in PMs).

EM_f: Effort multiplier for the factor f.
In order to use all factors in such a calculation, another value called "the overall effort adjustment factor (EAF)" is used.

The formula of EAF is as follows:

COCOMO II

The COCOMO II method was developed using COCOMO-81 model. The model was developed by analyzing the changes in software engineering over the past 20 years and reflecting these changes. This model suits well for cost estimation of the modern software projects.

COCOMO II provides two main models:

A. Early Design Model
B. Post-architecture Model

A. Early Design Model

In the requirements analysis phase of a software project's lifecycle, the stakeholders must agree upon the requirements. The Early Design Model gets into action when these requirements get agreed upon. The estimated effort required for the software project is calculated using the following formula:

$$\text{Effort}_{NS} = A \times (\text{Size})^{E} \times M$$

Where:

A: Constant (based on the calibration of local conditions and past data of the firm).
Size: Size of the software (expressed in KLOCs).
E: Constant
M: Constant (based on the attributes/cost-drivers of the project.
Effort_{NS}: Estimated effort (expressed in units of PMs).

The amount of the development time is calculated using the following formula:

$$\text{Time}_{dev} = C \times (\text{Effort})^{F}$$

Where:

C: Constant (based on the calibration of local conditions and past data of the firm).
F: Constant
Effort: Previously calculated estimated effort (expressed in units of PMs).
Time_{dev}: Estimated development time (expressed in months).

The constant E used in the calculation of the estimated effort and the constant F used in the calculation of the estimated development time are values that must be derived using formulas. Differently from the COCOMO-81, in the COCOMO II, the software projects are not categorized as organic, semi-detached or embedded types. Instead, there are *Scaling Factors* (SFs), which are used in determining the constants E and F. In COCOMO II, there are five different SFs that are listed below. Each of these factors is rated in 6 levels ranging between "very low" to "extra high".

Table 1: Scale factors

SFs	Very Low	Low	Nominal	High	Very High	Extra High
PREC (SF_1)	6.20	4.96	3.72	2.48	1.24	0.00
FLEX (SF_2)	5.07	4.05	3.04	2.03	1.01	0.00
RESL (SF_3)	7.07	5.65	4.24	2.83	1.41	0.00
TEAM (SF_4)	5.48	4.38	3.29	2.19	1.10	0.00
PMAT (SF_5)	7.80	6.24	4.68	3.12	1.56	0.00

The values of these weights are calculated based on statistics of a large number of software projects.The constants E and F are calculated using the formula:

$$E = B + .01 * \sum_{i}^{5} SFi$$

Where:

B: Constant (varies from 1.1 to 1.24 with respect to the novelty of the project, development flexibility, risk management methods and the process maturity).
SF_i: i^{th} scale factor weight.
E: Constant

$$F = D + 0.2 \times (E - B)$$

Where:

D: Constant (based on the calibration of local conditions and past data of the firm and is taken as 0.28 in the initial calibration).

E: Constant calculated using the previous formula.

B: Constant (varies from 1.1 to 1.24 with respect to the novelty of the project, development flexibility, risk management methods and the process maturity).

F: Constant (used in calculation of the amount of the development time).

The constant M that is used in calculating the estimated effort of a software project must be determined.

$$M = (PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED)$$

Where:

PERS, RCPX, RUSE, PDIF, PREX, FCIL AND SCED: Constant values of the properties of the Early Design Model.

B. Post-architecture Model

In the first phases of a software project's lifecycle, the architecture of the whole lifecycle is developed that defines various aspects about the project in details. The Post architecture Model was introduced to be used after the project lifecycle architecture is developed. In the Post-architecture Model, the estimated effort for the development of the software project is calculated with the same formula that is used in the Early Design Model. However, the Post-architecture Model uses 17 properties for this calculation instead of 7 properties used by the Early Design Model.

Product Attributes

- RELY (Required system Reliability)
- CPLX (Complexity of the System)
- DOCU(Documentation Required)
- DATA(Size of database)
- RUSE (Requirement for reuse)

Computer Attributes

- TIME (Execution Time constraint)
- PVOL(Development Platform Volatility)
- STOR (Memory Constraints)

HR Attributes

- ACAP (capability of project analyst)
- PCON (Personnel continuity)
- PCAP (Programmer capability)
- PEXP(programmer domain Experience)
- AEXP (Analyst domain Experience)
- LTEX (Language and tool experience)

Project Attributes

- TOOL (Use of Software tools)
- SCED (Schedule compression)
- SITE (Extent of multisite working)

Each of these properties has a value associated with it called Effort Multiplier (EM).

In this model, the constant M is calculated using the following formula:

$$M=RELY \times CPLX \times DOCU \times DATA \times RUSE \times TIME \times PVOL \times STOR \times ACAP \times PCON \times PGAP \times PEXP \times AEXP \times LTEX \times TOOL \times SCED \times SITE)$$

COCOMO's well definition offers various advantages:

1. It is easy to adapt.
2. It is very understandable.
3. It provides more objective and repeatable estimations
4. It creates the possibility of calibrating the model to reflect any type of software development environment and thus, providing more accurate estimates.
5. Works on historical data and hence is more predictable and accurate.

Disadvantages:

1. COCOMO model ignores requirements and all documentation.
2. It ignores customer skills, cooperation, knowledge and other parameters.
3. It oversimplifies the impact of safety/security aspects.
4. It ignores hardware issues
5. It ignores personnel turnover levels
6. It is dependent on the amount of time spent in each phase.

Function Point Analysis Based Methods

Function Point Analysis (FPA) is an ISO recognized method to measure the functional size of an information system. The functional size reflects the amount of functionality that is relevant to and recognized by the user in the business. It is independent of the technology used to implement the system. The unit of measurement is "function points". So, FPA expresses the functional size of an information system in a number of function points. All software applications will have numerous elementary processes or independent processes to move data. Transactions (or elementary processes) that bring data from outside the application domain (or application boundary) to inside that application boundary are referred to as external inputs. Transactions (or elementary processes) that take data from a resting position (normally on a file) to outside the application domain (or application boundary) are referred as either an external outputs or external inquiries. Data at rest that is maintained by the application in question is classified as internal logical files. Data at rest that is maintained by another application in question is classified as external interface files.

Types of Function Point Counts:

Development Project Function Point Count

Function Points can be counted at all phases of a development project from requirements up to and including implementation. This type of count is associated with new development work. Frequently, this type of count is called a baseline function point count.

Enhancement Project Function Point Count

It is common to enhance software after it has been placed into production. This type of function point count tries to size enhancement projects. All production applications evolve over time. By tracking enhancement size and associated costs a historical database for your organization can be built. Additionally, it is important to understand how a Development project has changed over time.

Application Function Point Count

Application counts are done on existing production applications. This “baseline count” can be used with overall application metrics like total maintenance hours. This metric can be used to track maintenance hours per function point.

Productivity:

The definition of productivity is the output-input ratio within a time period with due consideration for quality.

Productivity = outputs/inputs (within a time period, quality considered)

The formula indicates that productivity can be improved by (1) by increasing outputs with the same inputs, (2) by decreasing inputs but maintaining the same outputs, or (3) by increasing outputs and decreasing inputs change the ratio favorably.

Software Productivity = Function Points / Inputs

Software productivity is defined as hours/function points or function points/hours. This is the average cost to develop software or the unit cost of software. Function Points are the output of the software development process. Function points are the unit of software. It is very important to understand that Function Points remain constant regardless who develops the software or what language the software is developed in. On the other hand, to accurately estimate the cost of an application each component cost needs to be estimated.

- Determine type of function point count
- Determine the application boundary
- Identify and rate transactional function types to determine their contribution to the unadjusted function point count.
- Identify and rate data function types to determine their contribution to the unadjusted function point count.
- Determine the value adjustment factor (VAF)
- Calculate the adjusted function point count.

Function Point calculation

The function point method was originally developed by Bij Albrecht. A function point is a rough estimate of a unit of delivered functionality of a software project. Function points (FP) measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories:

Number of user inputs: Each user input that provides distinct application oriented data to the software is counted.

Number of user outputs: Each user output that provides application oriented information to the user is counted. In this context "output" refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

Number of user inquiries: An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files: Each logical master file is counted.

Number of external interfaces: All machine-readable interfaces that are used to transmit information to another system are counted.

Once this data has been collected, a complexity rating is associated with each count:

TABLE 2: Function point complexity weights.

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

Each count is multiplied by its corresponding complexity weight and results are summed to provide UFC. Adjusted function point count (FP) is calculated by multiplying the UFC by a Technical Complexity factor (TCF).

Table 3. Components of the technical complexity factor.

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

Each component is rated from 0 to 5, where 0 means the component has no influence and 5 means the component is essential (Pressman, 1997).

$$VAF=0.65+ (Sum (Fi))$$

$$Final\ Adjusted\ FP=UFC*VAF$$

$$EFFORT = EAF \times A \times (SLOC)^{EX}$$

$$EAF = CPLX \times TOOL$$

A = 3.2= Constant based on the development mode.
EX = 0.38= Constant based on the development mode.
CPLX = 1.3 = Constant based on the development language.
TOOL = 1.1 = Constant based on the development Tool.
TDEV = 2.5 x (EFFORT) EX in months

$$TDEV = 2.5 \times (EFFORT)^{EX} \text{ in months}$$

Putnam Model

This is analytical model which works on global assumptions [24]. The tool which implements Putnam model is SLIM. SLIM is based on the work of Norden/ Rayleigh [26] manpower distribution curves. The equation which implements Putnam model is called 'software equation'; it is of the form,

$$S = C_k * (Effort)^{1/3} * t_d^{4/3} \quad (1)$$

Ck is technology constant which shows the effect of language and development platform on effort.

$$Effort = D_0 * (t_d)^3 \quad (2)$$

Where D_0 is the magnitude of the difficulty gradient or manpower-build up From 1 and 2.

$$Effort = \left(D_0^{4/7} * E^{-9/7}\right) * S^{9/7}$$

$$DevTime = \left(D_0^{4/7} * E^{-3/7}\right) * S^{3/7}$$

There are other models such as ***Price-S*** [25] (Programming Review of Information Costing and Evaluation-Software) model which was developed by RCA PRICE systems. Like Putnam SLIM [24] the underlying concepts of the model is not publicly available but the important thing is that US Department of Defense demands a PRICE estimate for all quotations for a software project.

Non- Algorithmic methods

1. Computational Intelligence based: In recent years, Machine learning methods such as Artificial Neural Network, Genetic Algorithm are used in the prediction of software effort. These techniques reflect some of the functions of the human mind to solve highly complex problems. So they are called computational Intelligence Tools.

Neural Networks:

A *NN* is generally depicted on the basis of learning rules, characteristics adopted by neuron (nodes) and net topology [32]. A *NN* can be comprehended as an interconnected collection of artificial neurons that uses a mathematical or computational model for information processing. *NN* is a non-linear approach to deal with the complex behavior among inputs and outputs. Logically the neurons (nodes, Processing elements or units) are connected together to form a network. Each neuron is connected with the other by a connection link. Each connection link is associated with weights which contain information about the input signal. This information is used by the neuron net to solve a particular problem. Each neuron has an internal state of its own. This internal state is called the activation level of neuron, which is the function of the inputs the neuron receives. There are a number of activation functions that can be applied over net input such as Gaussian, Linear, Sigmoid and Tanh. Figure 4 shows the structure of a basic neural network.

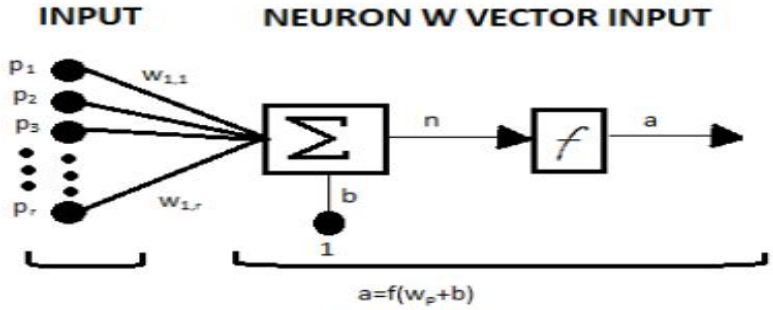

Fig4: Basic Neural Network

A basic neural network consists of a number of inputs applied by some weights, combined together to give an output. The feedback from the output is again put into the inputs to adjust the applied weights and to train the network. This structure of the neural networks help to solve the practical, non linear, decision making problems easily.

- Fuzzy Logics

  In 1965, Lofti Zadeh formally developed multi-valued set theory, and introduced the term fuzzy into the technical literature [7]. Fuzzy logic starts with the concept of fuzzy set theory. It is a theory of classes with un-sharp boundaries, and considered as an extension of the classical set theory.

The membership $\mu_a(x)$ of an element x of a classical set A, as subset of the universe X, is defined by:

$$\mu_A(x) = \begin{cases} 1 & if \quad x \in A \\ 0 & if \quad x \notin A \end{cases}$$

A system based on fuzzy logic has a direct relationship with fuzzy concepts (such as fuzzy sets, linguistic variables, etc.) and fuzzy logic. The popular fuzzy logic systems can be categorized into three types: pure fuzzy logic systems, Takagi and Sugeno's fuzzy system, and fuzzy logic system with fuzzifier and defuzzifier [7]. Since most of the engineering applications produce crisp data as input and expects crisp data as output, the last type is the most widely used one fuzzy logic system with fuzzifier and defuzzifier was first proposed by Mamdani. It has been successfully applied to a variety of industrial processes and consumer products [7].

The main fours components' functions are as follows:

- Fuzzifier: It converts a crisp input to a fuzzy set. ? Fuzzy Rule Base: Fuzzy logic systems use fuzzy IF-THEN rules.
- Fuzzy Inference Engine: Once all crisp input values are fuzzified into their respective linguistic values, the inference engine accesses the fuzzy rule base to derive linguistic values for the intermediate and the output linguistic variables.
- Defuzzifier: It converts fuzzy output into crisp output.
- Genetic Algorithms

Today's Software development effort estimation models are based on soft computing techniques as neural network, genetic algorithm, the fuzzy logic modeling etc. for finding the accurate predictive software development effort and time estimation. As there is no clear guideline for designing neural networks approach and also fuzzy approach is hard to use. Genetic Algorithm can offer some significant improvements in accuracy and has the potential to be a valid additional tool for software effort estimation. It is a non- parametric method since it does not make any assumption about the distribution of the data and derives equations according only to the fitted values. Genetic Algorithm is one of the evolutionary methods for the effort estimation. The solution is achieved by means of a cycle of generations of candidate solutions that are pruned by criteria „survival of the fittest [14]. Genetic Programming (GP) is a global search technique which makes it less likely to get stuck in the local optimum. This is different from other techniques such as neural networks and gradient descent which is prone to the local optimal values. It is particularly well suited for hard problems where little is known about the underlying search space and the concept is easy to understand [15]. The Genetic algorithm does combination of selection, crossover and mutation operators with the absolute aim of finding the most suitable solution to a estimation problem until actual criterion is met. Genetic algorithms start working with the initial population which evolves towards a population that is expected to produce good reasonable estimates. The solution to a problem in this domain is referred as a chromosome. This chromosome is usually an array of bits. These chromosomes are composed of a group of genes. The input estimation parameters are referred as genes in this structure. The reproduction evaluation for each iteration in Genetic algorithms is called generation. The probabilistic selection of chromosome from a current population is done based on the fitness function. Then the selected chromosomes are subjected to crossover, mutation and other operations.

Selection of Estimation Methods

Researchers have developed various techniques but no one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other.

Table4: Advantages and disadvantages of various estimation methods

S.No.	Method	Advantages	Disadvantages
1.	LOC	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	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.	Basic COCOMO	*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
4.	COCOMO II	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.

4.	Detailed COCOMO	**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.
5.	Expert Judgment	Fast prediction, Adapt to especial projects	Its success depend on expert, Usually is done incomplete
6.	Analogy	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
7.	Neural Networks	Consistent with unlike databases, Power of reasoning	There is no guideline for designing, The performance depends on large training data
8.	Fuzzy model	Training is not required, Flexibility	Hard to use, Maintaining the degree of meaningfulness is difficult
9.	Evolutionary Computation	It can search the vast space of possible combinations of cost drivers efficiently and reach a near-to-optimal outcome.	Gives heuristic solution

Conclusion

The accurate prediction of software development costs is a critical issue to make the good management decisions and accurately determining how much effort and time a project required for project managers as well as system analysts and developers. There are many software cost estimation methods available including algorithmic methods, estimating by analogy, expert judgment method, top-down method, and bottom-up method. No one method is necessarily better or worse than the other. For a specific project to be estimated, which estimation methods should be used depend on the environment of the project. According to the weaknesses and strengths of the methods, you can choose some methods to be used. For known projects and projects parts, we should use expert judgment method or analogy method if the similarities of them can be got, since it is fast and under these circumstance, reliable; For large, lesser known projects, it is better to use algorithmic model like COCOMO2.0 which will be available in early 1997. If COCOMO2.0 is not available, ESTIMACS or the other function point based methods are highly recommended especially in the early phase of the software life-cycle because in the early phase of software life-cycle SLOC based methods have great uncertainty values of size. If there are many great uncertainty values of size, reuse, cost drivers etc., the analogous method or wide-band Delphi technology should be considered as the first candidate. And , the COCOMO 2.0 has capabilities to deal with the current software process and is served as a framework for an extensive current data collection and analysis effort to further refine and calibrate the model's estimation capabilities. Therefore combination of estimation methods may generate reliable, accurate cost estimation for software development.

References

[1] B. Boehm, C. Abts, and S. Chulani, “Software development cost estimation approaches – A survey”, Ann. Softw. Eng., vol. 10, pp. 177-205, Jan. 2000.
[2] Y. F. Li, M. Xie, T. N. Goh, ”A Study of Genetic Algorithm for Project Selection for Analogy Based Software Cost Estimation, IEEE, 2007.
[3] B. W. Boehm, (1981.)Software engineering economics, Englewood Cliffs, NJ: Prentice-Hall.
[4] Sheta, A., Rine, D., & Ayesh, A. (2008, June). Development of software effort and schedule estimation models using soft computing techniques. InEvolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on (pp. 1283-1289). IEEE.
[5] Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. IEEE transactions on Software Engineering, 4(4), 345-361.
[6] Abbas, Syed Ali, et al. "Cost Estimation: A Survey of Well-known Historic Cost Estimation Techniques." Journal of Emerging Trends in Computing and Information Sciences 3.4 (2012).

[7] Ghose, Mrinal Kanti, Roheet Bhatnagar, and Vandana Bhattacharjee. "Comparing some neural network models for software development effort prediction." Emerging Trends and Applications in Computer Science (NCETACS), 2011 2nd National Conference on. IEEE, 2011.
[8] Kashyap, Divya, Ashish Tripathi, and A. K. Misra. "Software Development Effort and Cost Estimation: Neuro-Fuzzy Model." (2012).
[9] Vinay Kumar, K., et al. "Software development cost estimation using wavelet neural networks." Journal of Systems and Software 81.11 (2008): 1853-1867.
[10] Attarzadeh, Iman, and Siew Hock Ow. "Proposing a new software cost estimation model based on artificial neural networks." Computer Engineering and Technology (ICCET), 2010 2nd International Conference on. Vol. 3. IEEE, 2010.
[11] Khaled Hamdan, Hazem El Khatib, Khaled Shuaib," Practical Software Project Total Cost Estimation Methods", MCIT 10, IEEE, 2010.
[12] Alaa Sheta., (2006), "Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects", Journal of Computer Science 2 (2): 118-123.
[13] Sumeet Kaur Sehra, Yadwinder Singh Brar, Navdeep Kaur, "Soft Computing Techniques for Software project Effort Estimation" International Journal of Advanced Computer and Mathematical Sciences, Vol. 2, No. 3, 2011, ISSN 2230-9624, pp 162-163.
[14] Jin-Cherng Lin, Chu-Ting Chang, Sheng-Yu Huang, "Research on Software Effort Estimation Combined with Genetic Algorithm and Support Vector Regression" International Symposium on Computer Science and Society, In 2011 IEEE, pp. 349- 352
[15] K. Strike, K. El Emam, and N. Madhavji, "Software cost estimation with incomplete data", IEEE Trans. Softw. Eng., vol. 27, no. 10, pp. 890-908, Oct. 2001.
[16] Attarzadeh, Iman, and Siew Hock Ow. "Project management practices: Success versus failure." Information Technology, 2008. ITSim 2008. International Symposium on. Vol. 1. IEEE, 2008.
[17] Chetan Nagar, "Software efforts estimation using Use Case Point approach by increasing technical complexity and experience factors", IJCSE, ISSN:0975-3397, Vol.3 No.10 , Pg No 3337- 3345,October 2011.
[18] Urkola Leire , Dolado J. Javier , Fernandez Luis and Otero M. Carmen , (2002), "Software Effort Estimation: the Elusive Goal in Project Management", International Conference on Enterprise Information Systems, pp. 412-418.
[19] Chetan Nagar, Anurag Dixit, "Software efforts and cost estima- tion with systematic approach", IJETCIS, ISSN:2079-8407, Vol.2 No.7, July 2011.
[20] Narendra Sharma, Aman Bajpai, Mr. Ratnesh Litoriya ",A Comparison of software cost estimation methods: A Survey", The International Journal of Computer Science & Applications (TIJCSA), Volume 1, No. 3, May 2012.
[21] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," IEEE Transactions on Software Engineering, vol. 21, pp. 126-137, 1995.
[22] A. R. Venkatachalam, "Software Cost Estimation Using Artificial Neural Networks," Presented at 1993 International Joint Conference on Neural Networks, Nagoya, Japan, 1993.
[23] G. H. Subramanian, P. C. Pendharkar and M. Wallace, "An Empirical Study of the Effect of Complexity, Platform, and Program Type on Software Development Effort of Business Applications," Empirical Software Engineering, vol. 11, pp. 541-553, 2006.
[24] 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.
[25] R. E. Park, "PRICE S: The calculation within and why", Proceedings of ISPA Tenth Annual Conference, Brighton, England, July 1988.
[26] Norden and V. Peter. Useful tools for project management, management of production, M.K. Starr, Penguin Books, Inc., Baltmore, Md., 1970, pp. 71-101
[27] Bardsiri, A. K., & Hashemi, S. M. Software Effort Estimation: A Survey of Well-known Approaches.
[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.