International Journal of Progressive Sciences and Technologies (IJPSAT)
Vol. 2 No. 2 April. 2016, pp. 48-60.

http://ijpsat.ijsht-journals.org

Software Effort Estimation: A Comparative Analysis

Varshan Chowdary[1], Vamsi Krishna[2]

[1,2] Department of Computer science, Vellore Institute of Technology,
Vellore, India

ABSTRACT: Software is the most expensive element of virtually all computer based systems. For complex custom systems, a large effort estimation error can make the difference between profit and loss. Number of researchers had made their effort to produce different modeling techniques in last few decades. This paper is about the descriptive exploration of various techniques proposed in software estimation field. In this paper we present the findings of various researchers in their research papers that have utilized a various parametric and non parametric techniques on effort estimation in a simplified manner. So, all the effort estimation techniques placed at one place helps the other researchers to distinguish and analyse which method is used for the particular project.

KEYWORDS: Comparative analysis, Software effort estimation, algorithmic methods.

1. INTRODUCTION

The modern day software industry is all about efficiency. Day to day competition in the software industries is increasing; in such scenario accurate effort estimation has become an important task. In general software effort estimation techniques can be subdivided into experience based, parametric model based, learning-oriented, dynamics-based, regression-based and composite techniques. Amongst these methods, model-based techniques involve the use of mathematical equations [1]. Software cost and effort estimation is become a challenge for IT industries. There are many methods which was mentioned above, but most of the people don't know how to use those methods. Mainly IT industries are developing two types of projects. First type includes specifc customer; all requirements of the project are given by the customer. And in second type, in which requirements are gathered by using some survey. In case of second type effort estimation is more difficult because we don't have a specific customer and we have to estimate efforts accordingly and complete the project within time limit. So an effective estimation approach is required in this case. Accurate cost estimation is very important and needed to prioritize the development project and also to determine what resources are required to the project and how well these resources will be used. The effort prediction which is done at the early stages of the project is the best helpful one [2]. However, estimation at the early stages is most difficult because the primary source to estimate the costing comes from the requirement specification document. There are different competing methods of cost estimation available to the software developers to predict effort; from the expert opinion methods to the more complex algorithmic modeling methods and the analogy based methods [2], [11], [17]. Effort estimation is mainly classified into two algorithmic and non-algorithmic models. The algorithmic methods are largely studied and lot of models have been developed under this, like COCOMO model [3], Putnam model [5] etc. Each model in this paper are collected from several research papers and modified accordingly.

2. LITERATURE REVIEW:

Effort estimation is a key factor for software project success, defined as delivering software of agreed quality and functionality within schedule and budget. Traditionally effort estimation has been used for planning and tracking project resources. Effort estimation methods founded on those goals typically focus on providing exact estimates and usually do not support objectives that have recently become important within software industry, such as systematic and reliable analysis of causal effort dependencies. This paper presents the survey made by various professionals on various effort estimation methods. And this information is collected from the scholarly articles on various Effort Estimation methodologies.

Corresponding Author: Vamsi Krishna (bandivamsi95@gmail.com)

Software researchers and practitioners have been addressing the problem of effort estimation for software development projects since atleast the 1960's. Most of the rresearch has focused on the construction of formal software effort estimation models.The early models were typically based on regression analysis or mathematically derived from other domains. Since then a high number of model building approaches have been evaluated, such as approaches founded on case-based reasoning, classification and regression trees, simulation, neural networks, Bayesian statistics, lexical analysis of requirement specifications, genetic programming, linear programming, economic production models, soft computing, fuzzy logic modelling, statistical bootstrapping, and combinations of two more of these models. The perhaps most common estimation methods today are the parametric estimation models COCOMO, SEER-SEM and SLIM. They have their basis in estimation research conducted in the 1970s and 1980s and are since then updated with new calibration data, with the last major release being COCOMO II in the year 2000. The estimation approaches based on functionality-based size measures, e.g., function points, is also based on research conduced in the 1970s and 1980s, but are re-calibrated with modified size measures and different counting approaches, such as use case points in the 1990s and COSMIC in the 2000s.

The most common measure of the average estimation accuracy is the MMRE (Mean Magnitude of Relative Error), where the MRE of each estimate is defined as:

MRE=|actual effort-estimated effort|/actual effort

This measure has been criticized and there are several alternative measures, such as more symmetric measures, Weighted Mean of Quartiles of relative errors and Mean Variation from Estimate. A high estimation error cannot automatically be interpreted as an indicator of low estimation ability. Alternative, competing or complementing, reaasons include low cost control of project, high complexity of development work, and more delivered functionality than originally estimated. A framework for improved use and interpretation of estimation error measurement is included.

There are many psychological factors potentially explaining the strong tendency towards over-optimistic effort estimates that need to be dealt with to increase accuracy of effort estimates. These factors are essential even when using formal estimation models, because much of the input to these models is judgement based. Factors that have been demonstrated to be important are: Wishful thinking, anchoring, planning fallacy and cognitive dissonance. A discussion on these factors can be found in the work by Jorgensen and Grimstad.

1. It's easy to estimate what you know.
2. It's hard to etimate what you know you don't know.
3. It's very hard to estimate things that you don't know you don't know.

The chronic underestimation of development effort has led to the coinage and popularity of numerous humorous adages, such as ironically referring to a task as a "small matter of programming", and citing laws about underestimation:

a. Ninety-ninety rule:

The first 90% of the code accounts for the first 90 percent of the development time. The remaining 10% of the code accounts for the other 90% of the deelopment time.--**Tom Cargill, Bell Labs.**

b. Hofstadter's law:

It always takes longer than you expect, even when you take into account.—**Douglas Hofstadter, Godel Escher, Bach.**

3. VARIOUS EFFORT ESTIMATION METHODS:

3.1. Expert Judgement Method

In this method, project is assigned to a cost estimation expert or a group of experts to estimate the cost according to their understanding to the proposed project. We will get differing opinions with a group, so to co-ordinate the opinions there are some techniques, one of these techniques is Delphi. There are more different Delphi forms.

The advantages of this method are:

1.Experts can estimate the requirements of the present project by comparing the project with the past projects.

2.Experts can also analyse the impacts on the present project caused by technologies, languages, applications, architectures involved in the upcoming projects and estimate accordingly.

The disadvantages include:

1.Quantisation of this model is not possible.

2.Documentation of the factors used by group of experts or experts is hard.

3.Psychological issues of the expert may affects the estimation, eg.what if the expert is biased and optimistic?

3.2. Estimating by Anology

Analogy method is more frequently used method not only in cost estimation but also in daily life problems. In this method estimation of the present project is done by analysing similar previous projects and the method used in that situation.

The main advantages of this method are:

1.Estimation is done by analysing real time projects in the past. Past experience of the estimator is used.

2.Impacts also estimated by differing completed and present projects.

And the disadvantages are:

1.To use this method, description of the project must be clearly describes and the selection of variables also redtricted to the data at the point where prediction is required.

2.There also exist some situations which include the type of domain application runs, range of inputs and distinct referenced entities and so on.

3.3. Top-Down Estimating Method

This method is otherwise known as Macro Mode. Cost estimation in this method is done by using the global attributes of the project and then divided into lower-level components. Putnam model uses this approach which is one of the leading methods in present day situations. When only global attributes are known, this method is applicable for good results.

The advantages of this method are:

1. System-level activities such as documentation, integration, configuration management etc., are highly focussed by this method, which are mostly ignored in other methods of esimation and the cost of system level functions will not be missed by this method.

2. No need of whole project details, requires basic details, easier and faster implementation.

The disadvantages are:

1. It can’t identify the problems faced in lower levels which are likely to increase the cost.

2.It doesn’t provide any details for justifying estimates and decisions. As it only gives global view of the project, it embodies some efficient features which we see in putnam model.

3.4. Bottom-up Estimating Method

In this method estimation is done for all the components and combine all to get overall estimated cost of the project.

Estimation of the system in this method is done from the knowledge gained from the lower components and their interactions. COCOCMO model use this approach for estimation.

The advantages are:

1.It allows the software group in handling the estimate traditionally.

2.Estimation errors in various components can have a chance of balance out, so the estimate is more stable.

The disadvantages are:

1.Information which is necessary may not be available at early stages, so the method is inacurate.

2.It consume more time.

3.It may not get good estimated when time is limited.

3.5. Algorithmic Methods

Software estimation is done by using some mathematical equations and derivations in algorithmic models. The algorthmic method is designed to provide some mathemaical equations to perform software estimation. The mathematical equations derived are based on historical data and research and take inputs as Source Lines of code, range of functions used and other cost drivers such as risk assessments, skill-levels, language, design methodology,etc. These methods are largely studied and development of new models taken place such as COCOMO [3] ,putnam [5] and function points based models [6].

Advantages:

1.Generation of repeatable estimations is possible.

2.Customization of the formula is possible and also easy to data modification

3.It is efficient and able to support a family of estimations or a sensitivity analysis.

4.Bases on previous experience, it is calibrated objectively.

Diadvantages:

1.Exceptional conditions can't be dealt such as good teamwork and great match between tasks and skill-levels.

2.Any inaccuracy in cost driver will result in wrong estimation.

3.Quantization of factors and experience is difficult.

3.5.1. COCOMO

The constructive cost model which is otherwise known shortly as COCOMO. It is developed by Barry Boehm.It is one of the algorithmic software cost estimation models. This method uses a basic regression formula and also the parameters that are derived from the present and historical project features.

Three levels of the model is proposed by Boehm that are basic, intermediate, detailed.

1. The basic COCOMO model is a static, single-valued which computes software development effort as a function of size of the program which is expressed in estimated thousand delivered source instructions.

2. The intermediate COCOMO model computes software development effort as a function of size of the program and a set of fifteen "cost drivers" that include various subjective assessments of project attributes, hardware, product and personnel.

3. The advanced or detailed COCOMO model contains all characteristics similar to intermediate version and also the cost driver's impact on every step is also assessed.

COCOMO models are mainly dependent on the two equations:

1. Development effort: **MM = a * KDSI^b**

Based on MM - man-month / staff-month / person month is effort of one person in one month. There are 152 hours per Person month in COCOMO model.

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

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

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

3.5.1. Basic COCOMO

In the basic COCOMO applies the parameterised equation is applied without considering the detailed characteristics of the project.

$$MM = a * KDSI^{b}$$

$$TDEV = 2.5 * MM^{c}$$

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 constant for basic models

3.5.3. Intermediate COCOMO

For this model basic equation is used and the cost drivers which are fifteen in number are rated on a scale of 'very high' to 'very low' for the calculation of specific effort multiplier and each driver gives an adjustment factor that multiplies yields in the total EAF(Effort Adjustment Factor). If the cost driver is judged normal then the adjustment factor is 1.

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

In the intermediate COCOMO model, the estimate we get in the basic COCOMO model is multiplied with the effort multiplier for the estimation refinement.

$$\textbf{Effortf = Efforti × EMf}$$

Where;

Effortf: Refined effort value for the factor f (in PMs).

Efforti: Initial effort value calculated with the basic

COCOMO model (in PMs).

EMf: Effort multiplier for the factor f.

To use all factors in such a complicated calculation, another value called “the overall effort adjustment factor (EAF)” is used.

3.5.4. COCOMO II

This method is developed from the COCOMO-81 model. This method is developed by analysing the various changes in Software engineering in past 20-30 years and correcting any drawbacks. In modern day projects cost estimation is best done by this model.

COCOMO II provides two main models:

A. Early Design Model

B. Post-architecture Model

A. Early Design Model

In the lifecycle of project and in requirements analysis phase, requirements must be agreed by the stakeholders.

When the requirements are agreed upon then the early design model is used. And the effort estimation is done by using fllowing formula.

$$\textbf{Effort NS = A × (Size) E × M}$$

Where:

A: Constant (based on the calibration of local conditions and past data of the firm).

Size: Size of the software (in KLOCs).

E: Constant

M: Constant (based on the attributes/cost-drivers of the project)

Effort NS: Estimated effort (in units of PMs).

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

$$\textbf{Time dev = C × (Effort) F}$$

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 constants E and F used in calculation of Effort estimation and in estimated development time respectively are derived using formulae. The software projects are not divided as organic, semi-detached or embedded types in this model, unlike COCOMO-81. We can also use scaling factors (SFs) to determine E and F values. There are five different SFs in COCOMO-II, which are listed below. Every factor is rated in 6 levels ranging between “very low” to “extra high”.

Table 1. Scale factors

SF’s	Very low	Low	Nominal	High	Very high	Extra HIgh
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

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

Where:

D: Constant (depends on firm’s old data and local conditions and 0.28 is taken as initial calibration).

E: Constant calculated by using the former 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. This model is basically introduced to use after the architecture of project lifecycle. Effort estimation in this model for the project development is calculated using the formula used in Early Design Model. It uses 17 properties for this calculation instead of 7 properties used in 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×CPLX×DOCU×DATA×RUSE×TIME×PVOL×STOR×ACAP×PCON×PGAP×PEXP×AEXP×LTEX×TOOL×SCED×SITE)

COCOMO's advantages:

1. Adaption is easy.

2. Easy to understand.

3. More repeatable and objective estimations are provided by this model.

4. Calibration of model to reflect various software development environment types and also provide accurate estimates.

Disadvantages:

1. Ignores all documentation and requirements.

2. Ignores customer cooperation, knowledge, skills and other parameters.

3. Oversimplification of the impact of safety/security aspects.

4. Ignores hardware issues.

5. Ignores personnel turnover levels.

6. Dependent on time spent in every phase.

3.5.5. Function Point Analysis Based Methods

Function Point Analysis (FPA) is an ISO method to measure the functional size of an information system. The functional size is the amount of functionality that is equal to and used by the user in the business. It is not dependent of the technology used to implement the system. The measured unit is "function points". So, FPA expresses the functional size of an information system in a number of function points. There are many elementary processes in software applications to move data. The elementary processes that bring data from outside the application boundary to inside that application boundary are considered to as external inputs. Elementary processes that take data from a resting position to outside the application boundary are considered as an external outputs. Data at rest that is maintained by the application in question is considered as internal logical files. The data which is at rest that is maintained by another application in questions is considered 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. Generally, this count is called a baseline function point count. This count is associated with new development work.

Enhancement Project Function Point Count

It is general to improve software after it has been placed into production. This function point count tries to make enhancement projects. All the applications in production will evolve over time. Historical database can be created by measuring the enhancement size and project cost. Along with this, it is also important to understand how a development project has changed over time by time.

Application Function Point Count

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

Productivity:

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

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

The formula states that productivity can be improved by increasing outputs with the same inputs or by decreasing inputs but maintaining the same outputs, or by increasing outputs and decreasing inputs change the ratio favourably.

Software Productivity = Function Points / Inputs

Software productivity is defined as hours per function point or function point per hours. This is 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 and does not depend on who develops the software or language of the software developed in. On the other way, to accurately estimate the cost of an application we need to estimate the cost of each component.

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 is developed by Albrecht. A function point is a fair estimate of a unit of delivered functionality of a software project. Function points (FP) measure size in terms of functionality in a system. Function points are DONE 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 entries	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

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

Each component is rated from 0 to 5, where 0 is the component has no influence and 5 is the component is essential.

VAF=0.65 + (Sum (Fi))

Final Adjusted FP=UFC*VAF

Effort=EAF x A x (SLOC) ^EX

EAF=CPLX x 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.

3.5.6. Putnam Model

This is the model which works on global assumptions [18]. SLIM is the tool which implements Putnam model. SLIM is based on the work of Norden / Rayleigh [20] distribution curves. Putnam model is implemented by the equation 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 D0 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 namely models such as Price-S [19] (Programming Review of Information Costing and Evaluation-Software) model which was developed by RCA PRICE systems. Like Putnam SLIM [18] the 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.

3.6. Non-Algorithmic methods

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.

3.6.1. Neural Networks

A NN is generally represented on the basis of learning rules and the characteristics adopted by neuron (nodes) and net topology [15]. A NN can be said as an interconnected collection of artificial neurons that uses a mathematical method for information processing. NN is a non-linear approach to deal with the complexity among inputs and outputs. Generally, the neurons are in a group of network. Each neuron is interlinked. Each link comprises of weights which contain information about the input signal. This information is used by the neuron group to solve a particular problem. Every neuron has a specific internal state. This internal state is called the activation level of neuron, where the information the neuron receives. There are a number of activation functions that can be applied over group input such as Gaussian, Linear, Sigmoid and Tanh. Fig.4 shows the structure of a basis neural network.

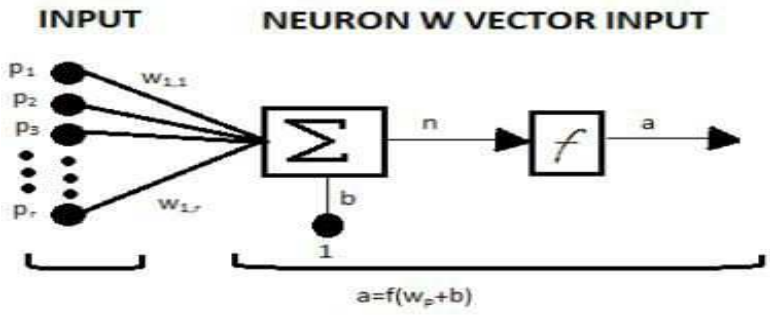

Fig. 3. Basic Neural Network

A neural network generally consists of a number of inputs applied b come weights, combined together to give an output. The data from the output is again given to the inputs to adjust the applied weights and to train the network. This construction of neural networks helps to solve non-linear decision making problems easily.

Today many Software development effort estimation models depend on soft computing techniques as neural network, genetic algorithm etc. to find the accurate predictive software development effort and time estimation. As there are no clear rules for designing neural networks approach and also fuzzy model is hard to use. By using genetic Algorithm we can get significant improvements in accuracy and can be potential to be a valid additional tool for software effort estimation. It is 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 best methods for the effort estimation. The solution is achieved by a cycle of generations of candidate solutions that are divided by criteria, survival of the fittest [14]. Genetic Programming is a best technique which makes it less likely to get struck in the local optimum. This is better than neural networks and gradient descent which depend on local optimum values. It is very well suited for hard problems where little is known about the underlying search space and concept is easy to understand [15]. The genetic algorithm is the combination of selection, crossover and mutation operators with the perfect aim of finding the most optimum 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. Chromosome is the solution of the problem in this domain. Generally chromosome is in array of bits. The chromosomes contain group of genes. The input parameters are considered as genes in this structure chromosome from a current population and is based on the fitness function. Then we use crossover mutation and other operations to find.

4. CONCLUSION

For a project to make more profit needs accurate effort estimation. There are many methods to estimate the cost including algorithmic methods, analogy based estimation, expert judgement method, top-down and bottom-up methods. But, no method is less considerable than the other as the estimation methods more dependent on project environment. If a project which is more similar and similar parts use expert judgement or analogy methods as it takes less time if the similarities are got; where the projects with lesser similarities and large in size uses algorithmic methods like COCOMO or for more precise cases COCOMO2.0. Function point based methods like ESTIMACS are used if COCOMO is not available. According to the merits and demerits of the method one have to choose the method keeping the project environment in mind. There are also some hybrid estimation methodologies like combination of Ant colony Optimization and Chaos Optimization Algorithms which serves better and more efficient than COCOMO model.

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. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on pp. 1283-1289.

[5] Putnam, L. H, “A general empirical solution to the macro software sizing and estimating problem”, IEEE transactions on Software Engineering, vol. 4, no. 4, pp. 345-361, 1978.

[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, vol. 3, no. 4, 2012).

[7] Ghose, Mrinal Kanti, Roheet Bhatnagar, and Vandana Bhattacharjee. "Comparing some neural network model 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] 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.

[19] R. E. Park, “PRICE S: The calculation within and why”, Proceedings of ISPA Tenth Annual Conference, Brighton, England, July 1988.

[20] Norden and V. Peter. Useful tools for project management, management of production, M.K. Starr, Penguin Books, Inc., Baltmore, Md., 1970, pp. 71-101.