Early Phase Software Effort Estimation Model: A Review

Priya Agrawal, Shraddha Kumar
CSE Department, Sushila Devi Bansal College of Technology, Indore(M.P.), India

***Abstract*—** *Software effort estimation is the measurement of work and resource required to develop a software system for a particular period. Cost can be determined using effort as cost is directly proportional to effort. Software cost estimation is one of the most difficult jobs in project planning especially in its early phase. In earlier proposed cost estimation models, cost estimation is done with more than 20 parameters at the early conceptual phase and if input is not defined using logical approach, the results of estimation are unpredictable. Using too many cost drivers or parameters make cost estimation process lengthy and complex. In our proposed study, we will highlight on the pros and cons of different cost estimation methods. Then we will look forward to work on making a simple system with minimum set of parameters that can be easily identified at an early stage while considering all possible aspects such as accuracy, simplicity etc.*

***Keywords*—** *Project Estimation, Effort Estimation, Cost Models, KSLOC, SDLC, Early Phase Cost Estimation*

I. INTRODUCTION

Cost estimation is the expert measurement of effort and development time required to develop a software system. The software cost estimation process includes determining the size of the software product to be produced, determining the effort required, developing initial schedules of projects, and finally, calculating the overall cost of the project. Software cost estimation requires the determination of the following estimates:

- Effort (usually in person-months)
- Project Duration (in calendar time)
- Number of persons required

After the large acceptance and success of COCOMO II model [4] of Boehm since 2000, there has not been much work done in this field for further improvement. An estimation model with 20-30 input parameters for cost drivers is not very helpful if we do not have a logical approach for specifying the input values for parameters like the software's complexity, database size, platform volatility, schedule compression, or personnel experience. After many years of COCOMO II an innovative approach came was of Wilson Rosa et al "Simple Empirical Software Effort Estimation Model" , 2014 [17] . His model overcomes the problem of having too many input parameters as in COCOMO II.

In this research investigation, we will understand the issues and challenges related to software development effort estimation. Cost estimation must be accurate because it is very helpful in determining what resources to commit to the project and how well these resources are used. Customers expect that actual project development cost to be in line with estimated costs, this gives better customer satisfaction.

II. LITERATURE SURVEY

Software Cost Estimation (SCE) is a process of forecasting of efforts and cost in terms of cost, schedule and personnel for any software system. Software cost estimation is a method which is as old as the computer industry itself and it has been developed many times until function points were formulated by Albrecht in 1979. Nowadays software cost estimation is becoming a complex branch of computer science, therefore many sizing techniques, sizing metrics, cost and effort models appeared. The author Abedallah Zaid et.al, [12] shows the common techniques used in SCE and along with it, it highlights the very important trends in this field also. He described the most urgent topics to investigate and the challenges in SCE.

Software project planning is one of the most important tasks in software project development. Poor planning of software development often leads to many problems in the long term of project use. Project errors & unwanted and unrealistic outputs are some common problems often occur in front of the project team. Nowadays software project managers should be aware of the increasing cases of project failures. The reason behind it is imprecision of the cost estimation. The author Vahid Khatibi et.al, [13] shows several existing methods for software cost estimation and demonstrated their aspects. Comparing the characteristics of the methods they can be applied for ability based clustering; it is also helpful in selecting the special method specific for each project. The author also gives a complete case study of estimation in an actual software project.

Software cost estimation is a critical factor in project management. If we fail to use right software cost estimation method, it might become a reason for the project failures. According to Report found by author Ali Bou Nassif et.al. [14], approximate 65% of

projects are delivered to client over budget or post-delivery deadline. Performing SCE in the early phases of the SDLC is crucial and this would be helpful to project development managers to bid on projects. The authors proposed a novel method to determine a software effort estimation using a cascade correlation neural network approach on Use Case Diagrams. They evaluated results based on the criteria of the MMER (Mean Magnitude of Error Relative to the estimate) and PRED (Prediction Level). They use 214 industrial projects and 26 educational projects to test the results on Use Case Point model and multiple linear regression models. The author concludes that the proposed model can be positively used with acceptable results as an alternative approach to calculate software effort as an early design phase of software development.

Most of the estimation models require details of different cost drivers that will be available at the later stage of the development process. The author Tharwon Arnuphaptrairong [15] proposes to use Function Point Analysis in application with a dataflow diagram to solve the timing critical problem. The proposed methodology by the author was validated through the graduate students' software. Although the results got by author were disappointing, but some interesting insights are worth looking at the model.

The author Wilson Rosa et.al. [17] explained that an SCE method with so many parameters that cannot be defined logically for input is not useful at an early conceptual stage. Author gives a simple approach for forecasting software development effort at an early stage of project development. The regression model is used along with product size and application types to calculate effort in this approach. Product size is calculated in terms of the equivalent source lines of code. The author gathered and then analyzed empirical data from 317 very recent projects implemented within the US Department of Defense over the tenure of 9 years started in 2004. The equation is easier and more relevant to use for early cost estimation than traditional parametric cost models. The Statistical results explained that source lines of code and application type, both are important contributors to the development effort.

III. ESTIMATION TECHNIQUES

Usually software cost estimation methods are divided into two classes: Algorithmic and Non Algorithmic. Both the classes are required for performing accurate estimation. In this section, some popular estimation techniques are discussed.

A. Algorithmic Models

This software cost estimation technique uses mathematical equations to perform the software estimation. The mathematical equations are based on previous data. SLOC (source line of code), function points, and other cost drivers are the inputs. For most algorithmic models, calibration to the specific software environment can be performed to improve the estimation.

1) Function Point Analysis: It starts with the breaking up of a project or application into its transactional functions and data. The data functions display the functionality provided to the user by fulfilling their internal and external needs in correlation to the data, whereas the transactional functions describe the functionality provided to the user in relation to the processing this data by the application. Each function is divided in accordance to its functional complexity as low, average or high. The data functions relative functional complexity depends on the type of of data element and the number of record element types (RETs). The transactional functions are classified according to the number of file types referenced (FTRs) and the number of DETs. The number of sum of the number of ILFs and the number of EIFs updated or queried during an elementary process.

The data functions are:
i.Internal Logical File (ILF)
ii.External Interface File (EIF)

The transactional functions are:
i.External Input (EI)
ii.External Output (EO)
iii.External Inquiry (EI)

The actual calculation process consists of three steps:
i.Determination of unadjusted function points (UFP).
ii.Calculation of value of adjustment factor (VAF).
iii.Calculation of final adjusted functional points.

Calculation of Unadjusted FP: The unadjusted Functional points are evaluated in the following manner:

$$UFP=\Sigma\Sigma Fxy*Zxy, \text{ for } y= 1 \text{ to } 3 \text{ and } x = 1 \text{ to } 5,$$

Where, Zxy denotes count for component x at level (low, average or high) y, and Fxy is corresponding Function Points.

Evaluation of Value Adjusted FP: Value Adjustment Factor (VAF) is evaluated from the addition of the degree of influence (DI) of the 14 general system characteristics (GSCc). General System characteristics are:

i. Data communications
ii. Distributed data processing
iii. Performance
iv. Heavily utilized configuration
v. Transaction rate
vi. On-line data entry
vii. End-user efficiency
viii. On-line update

ix. Complex processing
x. Reusability
xi. Installations ease
xii. Operational ease
xiii. Multiple sites/organizations
xiv. Facilitate change

Function points can be changed into Effort in Person Hours. Numbers of studies have attempted to relate LOC and FP metrics. Historical data for numerous programming languages derives the average number of source code statements per function point. Languages are classified into different levels according to the relationship between Lines Of Code and Function Point.

2) *KSLOC:* Source a line of code (SLOC) is software metric used to measure the size of a software program by counting the lines in the the program's source code. SLOC is typically used to predict the amount of effort that is needed to create a program, and also to estimate programming productivity or maintainability once the software is produced. SLOC is an estimation parameter that illustrates the number of all commands and data definition but comments and blanks are not considered in it. After completing the project, all estimations are verified with the actual ones. Thousand Lines of Code are used for estimation very commonly. SLOC calculation is very difficult at the early phase of the project because of the lack of information about requirements.

Since SLOC is measured based on language instructions, comparing the size of software which uses varied languages is too hard. However, SLOC is the foundation of the estimation models in many complicated software estimation methods. SLOC is measured by considering SL as the lowest, SH as the highest and SM as the most probable size.

$$S = (SL+ 4SM+ SH) / 6$$

3)COCOMO-II: The Early phase cost estimation model COCOMO-II uses thousand source lines of code (KSLOC) or unadjusted function points (UFP) for the estimation of size. UFPs can be changed to the equivalent SLOC and then to KSLOC to estimate the size of the software. The use of exponential scale factors is similar for Post-Architecture and the Early Design models. A reduced set of multiple cost drivers is used in the Early Phase Design model as shown in Table I. The Early Design phase cost drivers are established by integrating the Post-Architecture model cost drivers. The value of the cost drivers is calculated and whenever it lies in the midway of the rating provided, roundup this to the nominal rating. Example: If the rating of estimated cost drivers value lies between Very High and High then select High. The effort equation is same except that the number of effort multipliers is reduced to seven (n = 7).

TABLE I
EARLY DESIGN AND POST-ARCHITECTURE EFFORT MULTIPLIERS [8]

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
PERS	ACAP, PCAP, PCON
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

The reduced Early Design model cost driver maps all the Post-Architecture cost drivers as shown in TABLE I. The details of the cost drivers are as follows:

- Personnel Capability (PERS) includes Analyst Capability (ACAP), Programmer's Capability (PCAP), and Personnel Continuity (PCON).
- Product Reliability and Complexity (RCPX) includes Required Software Reliability (RELY), Database size (DATA), Product Complexity (CPLX), and Documentation match to life-cycle needs (DOCU).
- Required Reuse (RUSE) is same in both the phase of software development.
- Platform Difficulty (PDIF) includes Execution Time (TIME), Main Storage Constraint (STOR), and Platform Volatility (PVOL).
- Personnel Experience (PREX) includes Application Experience (AEXP), Platform Experience (PEXP), and Language and Tool Experience (LTEX).
- Facilities (FCIL) include use of Software Tools (TOOL) and Multisite Development (SITE).
- Schedule (SCED) is same in both the phase of software development.

The reduced cost drivers include the combination and use of numerical equivalent rating level values of complete cost drivers. Numerical values of Post-Architecture cost driver rating is 1 for Very Low, 2 for Low, 3 for Nominal, 4 for High, 5 for Very High and 6 for Extra High. The Early Design model cost drivers rating scale is from Extra Low to Extra High and the values of each cost driver is computed by the summation of values of combined Post-Architecture cost drivers value . If the contributing Post-Architecture cost driver has Nominal scale than the corresponding Early Design model rating is also Nominal. Effort will be calculated after analysis of each cost drivers value with the following given equation:

$$PM_{Estimated} = 3.67 \times (Size)^{(SF)_i} \times [\prod EM_i]$$

Where PM gives effort in persons-month, size in terms of KSLOC of the software, SF is the scaling factor and EM is the effort multipliers.

Mainstream parametric cost models use source lines of code (SLOC) as a measurement for predicting software effort. The main reason of using SLOC is that it allows practitioners to determine or at least

estimate what parts of the system they will actually be developing. In contrast, function-point or use-case based size estimates are made without determining which functions are going to be provided by Commercial-Off-The-Shelf products, cloud services, or other non-developed items, causing serious overestimates. However controversy exists over whether or not SLOC is a good indicator, continuous use of this metric generates meaningful statistical results.

4) Empirical Software Effort Estimation: An effort estimation model with more than 15 cost drivers is not very good at early conceptual phase if you do not have a logical approach for specifying the input values. This model presents a simple approach for predicting software development effort. The model uses software size and application types to estimate effort. Software size is evaluated in terms of the equivalent source lines of code. This study is based on empirical data assembled from 317 very recent projects developed within the United States Department of Defense over the period of nine years starting from 2004. Statistical results present that source lines of code and application type are important part of development effort. In comparison to traditional cost models, the equation is easier and more feasible to use for early estimates. Application type is defined in terms of different type of applications that can be developed shown in Table II.

$$PM = (2.047 \times KESLOC0.9288) \times (2.209^{D1}) \times (1.917^{D2}) \times (3.068^{D3}) \times (3.072^{D4}) \times (3.434^{D5}) \times (4.521^{D6}) \times (4.801^{D7}) \times (4.935^{D8}) \times (5.903^{D9}) \times (7.434^{D10}) \times (10.72^{D11})$$

Where:

PM = Engineering Effort for application type in Person Months

KESLOC = Product size in thousand Equivalent Source Lines of Code

D1 to D11 defines the application type. The values of each from D1 to D11 are 0 or 1 depending on the type of application to be developed. Table 2.2 shows the application type for all D1 to D11.

TABLE II
APPLICATION TYPE TAXONOMY [17]

Application Type	**Symbol**	**SEER-SEM Application Domain(s)**
Test	TST (D6)	Diagnostics, Testing Software
Software Tools	TUL (D1)	Business Analysis Tool, CAD, Software Development Tools
Intelligence & Information Systems	IIS (D2)	Database, Data Mining, Data Warehousing, Financial Transactions, GUI, MIS, Multimedia, Relational/Object-Oriented Database, Transaction Processing, Internet Server Applet, *Report Generation*, Office Automation
Mission Planning	PLN (D1)	Mission Planning & Analysis
Mission Processing	MP (D8)	Command/Control
Real Time Embedded	RTE (D7)	Embedded Electronics/Appliance, GUI (cockpit displays), Robotics
Scientific Systems	SCI (D3)	Expert System, Math & Complex Algorithms, Simulation, Graphics
Sensor Control and Signal Processing	SCP (D11)	Radar, Signal Processing
System Software	SYS (D4)	Device Driver, System & Device Utilities, Operating System
Telecommunications	TEL (D5)	Communications, Message Switching
Vehicle Control	VC (D9)	Flight Systems (Controls), Executive
Vehicle Payload	VP (D10)	Flight Systems (Payload)

B. Non Algorithmic Methods

In opposition to the Algorithmic methods, non algorithmic methods are based on analytical comparisons. In this method historical data is used which comes from similar project type and generally estimation is done according to the study of previous data. Here, three methods have been discussed which are popular.

1) Analogy: Some similar completed software projects are selected from previous database and then cost & effort estimation of under estimate project is done according to actual cost and effort of that projects. Estimation through analogy is proficient at the total structure levels and substructure levels. By judging the results of previous actual projects, we can estimate the cost and effort of a similar project. Following are the steps of this method:

i) Selection of analogy
ii) Examining similarities and differences .
iii) Investigating of quality of analogy
iv) Doing estimation

A similarity function is being used in this method that compares features of two projects. There are two common similarity function used namely Euclidean similarity (ES) and Manhattan similarity (MS)

2) Expert judgment: In this method Estimation is performed by taking advices from experts who have good experiences in similar projects. This method is generally helpful when there is lack of data and gathered requirements. The main issue with this method is that estimation is as good as expert experience. Delphi technique is one of the most common method that works according to expert judgment. In Delphi technique a meeting of project managers is called. They are then allowed to debate. The real data about the application is then mined from

these debates or discussions. Delphi includes some steps:
i. The convener gives an estimation form to each expert.
ii. Each expert presents his own estimation (without discussing with others)
iii. The convener gathers all forms and sums up them (including mean or median) on a form and ask experts to start another iteration.
iv. steps (ii-iii) are repeated until an approval is gained.

3) *Machine learning Models*: Most methods about software cost estimation use statistical techniques, which are not able to present logic and strong results. Machine learning approaches could be suitable in this area because they can increase the accuracy of estimation by training rules of estimation and repeating the run cycles. Machine learning methods can be divided into two main methods, which are explained below.

i) Neural Networks: This consists of many layers and each layer is made of several elements known as neurons. Neurons examine the weights defined for inputs and based on that produce the outputs. Output is the actual effort, which is the main aim of estimation. Back propagation neural network is the best choice for software estimation problem as it balance the weights by comparing the network outputs and actual results as well as training is done succesfully.

ii) *Fuzzy Method:* A Fuzzy systems tries to simulate human behavior and reasoning. There are many times that we find conditions are not clear and decision making is tough, in such scenario fuzzy systems are very effective. Fuzzy logic often focuses the data that gets ignored. Following are four steps in the fuzzy method:
Step 1: Fuzzification: to evaluate trapezoidal no. for the linguistic terms.
Step 2: to create the complexity matrix by determining a new linguistic term.
Stage 3: to find the productivity rate and try for the new linguistic terms.
Stage 4: Defuzzification: to find the effort needed to finish a task and to compare the subsisting method.
In the first step fuzzification has been done by scale factors, cost drivers and size . In step 2, principals of Cocomo are considered and defuzzification is accomplished to gain the effort.

IV COMPARISON OF THE ESTIMATION METHODS

At this section according to the previous presented subjects, it is possible to compare mentioned estimation methods based on advantages and disadvanta ges of them.This comparison could be useful for choosing an appropriate method in a particular project. On the other hand, selecting the estimation technique is done based on capabilities of methods and state of the project. Table IV shows a comparison of mentioned methods for estimation. For doing comparison, the popular existing estimation methods have been selected.

TABLE IV
COMPARISON OF CURRENT METHODS

Method	Type	Strengths	Weaknesses
COCOMO Model	Algorithmic	Universal Approach; More predictable and accurate	Much historical data is required; V
Function Point	Algorithmic	Language independent	Quite time consuming; Complex to use
Expert Judgement	Non-Algorithmic	Useful in absence of Quantified and empirical data	Estimate is only as good as Expert's opinion; Hard to do document the factors used by experts
Analogy	Non-Algorithmic	Based on actual project data	Impossible if no comparable project has been created in the past
Neural Networks	Non-Algorithmic	Consistent with unlike databases; Power of reasoning	Performance depends on large training data
Simple Empirical	Algorithmic	Equation is simpler; Very less input parameters are used	Not highly precise
Fuzzy	Non-Algorithmic	Flexibility; Training is not required	Very complex to use
KSLOC	Algorithmic	Ubiquitous Technique; Automation is possible	Language dependent

V CHALLENGES OF SOFTWARE COST ESTIMATION.

The estimation of effort can be done through COCOMO-II, but it uses many cost drivers for effort estimation and at early design phase it is difficult to define them logically. Another model, i.e., Simple Empirical Software Effort Estimation Model uses only application type for the estimation of effort hence it lacks with other important cost drivers that are necessary to consider for the calculation of effort. There is no such method exist that can improve the system's efficiency and accuracy and keep it updated with the current scenario. Hence a system is needed with a feature of learning capability.

Generalize Issues in Cost Estimation

- Defensible estimates are needed at the early conceptual phase of a software-intensive system's definition and development.
- Any cost stimation technique with 20-30 parameters is not very good if you do not have a defensible approach for specifying the inputs.
- As with the changing technology and increasing variety of application development, such a system is needed that can cope up with the changing environment and also improves itself with time through learning.

A model is required for the estimation of effort that will take minimum cost drivers and able to predict more accurate estimation and also the minimum cost drivers used by the proposed model should be logically defined at early design phase of software development. Software cost estimation process face many difficulties to get a proper and accurate estimate for many reasons, since software is a non physical thing and with such non physical thing estimation is not very easy in nature. One of the main difficulties faced with estimation process is the data availability, which is required for validating the correctness of any project models, metrics & sizing technique. The availability of data from real projects is scarce to verify validity of new models, estimation and sizing techniques methods. Many of the models and sizing techniques proposed are based on a very small amount of data; few methods for example have used more than 30 UML files to establish their effectiveness. The metrics, models and functional sizing techniques hence produced have low reliability and little evidence of accuracy to their credit. Another challenge in software cost estimation is that data may be very sensitive. A technique providing accurate results for a company in a country say A may produce far off estimation for a different company in another country B, this issue highlighted by Wieczorek and Ruhe [18]. Another major issue with software cost estimation is that once a model has been fine tuned for accuracy it requires data in early phase of software development which is hard to come by. One solution to this problem is to study large no of projects and produce a generalized benchmarking dataset as done by ISBSG. This dataset was created using many existing completed software projects.

Coming towards other difficulties in the cost estimation is the fast changing software industry. The entire software development environment is volatile, in terms of new tools , technologies used, better hardware speed, new programming languages are rapidly changing. There is need of a Estimation techniques which are adaptable to newer situations. This is important for SCE techniques to be consistent with these changing factors of industry.

Furthermore, in addition to the problem of data requirement, also the interdependency between the cost parameters and how each one of them affects the final output is very complex task.

Another challenge is lack of standard procedures and restrictions in software development and cost estimation in particular.

IV. CONCLUSIONS

Nowadays we are observing many projects going over budget and failing. This is due to faulty cost estimation. In this study, we have described how any single cost estimation model is not suitable for all projects. Each model has its own principles with are logically very different from others. It has become a difficult task for project managers to select one model type. In this review study, we have found that there is a need for one single software cost estimation technique that is simple enough, uses less input parameters and gives above average accuracy for all kinds of projects. Future works in this field may comprise of a hybrid model combining strengths of all different types of models (Algorithmic and Non-Algorithmic)

REFERENCES

[1] Boehm, Barry W., "A Technical Book on Software Engineering Economics." in Prentice Hall, 1981.

[2] Boehm B., "Software Engineering Economics" in Englewood Cliffs, NJ, Prentice---Hall, 1981

[3] Boehm B., Abts C., Brown W., Chulani S., Clark B., Horowitz E., Madachy R., Reifer D., Steece B., "Software Cost Estimation with COCOMO II", in Prentice---Hall, 2000

[4] Boehm, B., "Safe and simple software cost analysis," in Software, IEEE, 17(5), pp. 14–17, 2000.

[5] Clark, B., Devnani-Chulani, S., and Boehm, B., "Calibrating the COCOMO II Post-Architecture model," in Proc. Int'l Conf. Software Eng. (ICSE '98), pp. 477–480, 1998.

[6] Boehm, B., 2001 COCOMO Website: http://sunset.usc.edu/research/COCOMOII/cocomo_main.html

[7] Boehm, B., 2001 COCOMO Website: http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/help.html

[8] Boehm, B., "COCOMO II Model Definition Manual" in University of Southern California, 1999.

[9] Hareton Leung and Zhang Fan, "Software Cost Estimation", in The Hong Kong Polytechnic University, 1999

[10] Dr. N. Balaji, N. Shivakumar & V. Vignaraj Ananth, "Software Cost Estimation using Function Point with Non-Algorithmic Approach", in Global Journal of Computer Science and Technology Software & Data Engineering, Vol. 13 Issue 8, 2013.

[11] Ian Sommerville, "Software Cost Estimation", in Software Engineering Book, 6th edition, 2000

[12] Abedallah Zaid, Mohd Hasan Selamat, Abdual Azim Abd Ghani, Rodziah Atan and Tieng Wei Koh, "Issues in Software Cost Estimation", in International Journal of Computer Science and Network Security, 350 VOL.8 No.11, 2008.

[13] Vahid Khatibi, Dayang N. A. Jawawi, "Software Cost Estimation Methods: A Review", in Journal of Emerging Trends in Computing and Information Sciences, Vol 2, 2011.

[14] Ali Bou Nassif, Luiz Fernando Capretz and Danny Ho, "Software Effort Estimation in the Early Stages of the Software Life Cycle Using a Cascade Correlation Neural Network Model", in International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2012

[15] Tharwon Arnuphaptrairong, "Early Stage Software Effort Estimation Using Function Point Analysis: Empirical Evidence", in IMECS Vol-2, 2013

[16] Brad Clark, Wilson Rosa, Barry Boehm and Ray Madachy, "Simple Empirical Software Effort Estimation Models", in 29th International Forum on COCOMO and Systems/Software Cost Modeling, 2014

[17] Wilson Rosa, Ray Madachy, Barry Boehm and Brad Clark, "Simple Empirical Software Effort Estimation Model", in ESEM'14, ACM, 2014.

[18] I Wieczorek, M Ruhe," How valuable is company specific data

Compared to multi-company data forsoftwareestimation?"METRICS.02 , IEEE, pp 237- 246, 2002