Volume 3, Issue 8, August 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Software Project Effort and Cost Estimation Techniques

Jyoti G. Borade
Department of Computer Science and Engineering GIT,
Chiplun ,Maharashtra, India.
jyoti.borade81@gmail.com
Vikas R. Khalkar
Department of Mechanical Engineering
GIT, Chiplun ,Maharashtra, India.
vikaskhalkar@rediffmail.com

Abstract—The main goal of software project cost and effort estimation is to scientifically estimate the required workload and its corresponding costs in the life cycle of software system. Software cost estimation is a complex activity that requires knowledge of a number of key attributes that affect the outcomes of software projects, both individually and in concert. The most critical problem is the lot of data is needed, which is often impossible to get in needed quantities. Hence, Software cost and effort estimation has become a challenge for IT industries. In this paper, several existing methods for software project effort, cost estimation are illustrated and their aspects are discussed. Also, it describes software metrics used for software project cost estimation. This paper summarizes existing literature on software project cost estimation. The paper includes comment on the performance of the estimation models and description of research trends in software cost estimation.

Keywords: KLOC (Kilo Line of Code), FP (Function Point), FPA (Function Point Analysis), UCP (Use Case Point), TP (Test Point), Object Point, COCOMO, Cost Estimation, Efforts Estimation, Person-Month, Person- Hours, Bayesian Belief Network.

I. Introduction

Software has played an increasingly important role in sys- tems acquisition, engineering and development, particularly for large, complex systems. For such systems, accurate es- timates of the software costs are a critical part of effective program management. The bulk of the cost of software devel- opment is due to the human effort and most cost estimation methods focus on this aspect and give estimates in terms of person-months. Accurate cost estimates are critical to both developers and customers. They can be used for request for proposal, contract negotiations, scheduling, monitoring and control. Underestimating the costs may result in management approving proposed systems that then exceed their budgets, with underdeveloped functions and poor quality, and failure to complete on time. Overestimating may result in too many re- sources committed to the projects, or, during contract bidding, result is not winning the contract, which can lead to loss of jobs. Software project failures have been an important subject in the last decade. Software projects usually don't fail during the implementation and most project fails are related to the planning and estimation steps. Hence accurate cost estimation is become a challenge for IT industries.

There are a 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 [3], [11], [17], [19]. In software project estimation, it is important to balance the relationships between effort, schedule and quality, which form the three essential aspects of the famous "magic" triangle. It is widely accepted that simply estimating one of these aspects without considering the others will result in unrealistic estimations. Classical estimation models are established based on linear or non-linear regression analysis, which incorporate fixed input factors and fixed outputs. The size of the project determining the scope is modeled as a main input of such models [2], [4], [12], [18]. One representative of such models is COCOMO [1], [5], [16]. The most critical problem in such an approach is the wealth of data that is needed to get regression parameters, which is often impossible to get in needed quantities. It really limits their usage for project estimation. In recent years, a flexible and competitive method, Bayesian Belief Network (BBN), has been proposed for software project estimation [8], [9], [10], [14].

Testing is an important activity to ensure software quality. Big organizations can have several development teams with their products being tested by overloaded test teams. In such situations, test team managers must be able to properly plan their schedules and resources. Also, estimates for the required test execution effort can be an additional criterion for test selection, since effort may be restrictive in practice. A good test execution effort estimation approach can benefit both tester managers and software projects. There is estimation model and an experience-based

approach for test execution effort [6], [7], [15].

The main objective of this paper is demonstrating the abilities of the software cost estimation methods and clustering them based on their features which makes helps readers to bet- ter understanding. The full paper is organized as follows: sec- tion II, after Introduction, discusses some literature reviews on software metrics. Section III discusses estimation techniques and project models. Section four includes the comparison of existing methods. An actual example is presented in section five and finally, the conclusion is illustrated in section six.

II. SOFTWARE METRICS

This section provides some background information on software conventional metrics used in software cost and effort estimation [4], [20].

A. Size Oriented Metrics:

i) Source Lines of Code (SLOC): is software metric used to measure the size of software program by counting the number of lines in the text of the program's source code. This metric doesnot count blank lines, comment lines, and library. SLOC measures are programming language dependent. They cannot easily accommodate nonprocedural languages. SLOC also can be used to measure others, such as errors/KLOC, defects/KLOC, pages of documentation/KLOC, cost/KLOC.

ii) Deliverable Source Instruction (DSI): is similar to SLOC. The difference between DSI and SLOC is that "if-then-else" statement, it would be counted as one SLOC but might be counted as several DSI [5].

B. Function Oriented Metrics:

Function Point (FP): FP defined by Allan Albrecht at IBM in 1979, is a unit of measurement to express the amount software functionality [5]. Function point analysis (FPA) is the method of measuring the size of software. The advantage is that it can avoid source code error when selecting different programming languages. FP is programming language independent, making ideal for applications using conventional and nonprocedural languages. It is base on data that are more likely to be known early in the evolution of project. Function types are as:

- **External** Inputs (EI) : it originates from user or transmit- ted from another application.
- **External** Outputs (EO) : it is derived data within appli- cation that provides information to the user.
- **External** Enquiries (EQ) : it is online i/p that results in the generation of some immediate s/w response in the form of an online output.
- **Internal** Logical Files (ILF) : is logical grouping of data that resides within the applications boundary and maintained via EI.
- **External** Interface Files (EIF) : is logical grouping of data that resides external to application but provides information that may be of use to the application.

Steps for calculating FP:

i) Unadjusted FP Count (UFP): Function types are ranked according to their complexity: Low, Average or High, using a set of prescriptive standards. Table I describes weight value of function type according to complexity [18]. After classifying each of the five function types,

TABLE I
WEIGHT VALUE BY FUNCTIONAL COMPLEXITY.

Function Type	Low	Average	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

TABLE II
UFP CALCULATION.

Function Type	Functional Complexity	Complexity Totals	Function Type Totals
EI	low average high	*3=— *4=— *6=—	—
EO	low average high	*4=— *5=— *7=—	—
EQ	low average high	*3=— *4=— *6=—	—
ILF	low average high	*7=— *10=— *15=—	—
EIF	low average high	*5=— *7=— *10=—	—
'Total Unadjusted Function Point Count'=			————

ii) Total Degree of Influence (TDI): This step involves assessing the environment and processing complexity of the project or application as a whole. In this step, the impact of 14 general system characteristics is rated on a scale from 0 to 5 in terms of their likely effect on the project or application as given below:

- 0 = No Influence
- 1 = Incidental
- 2 = Moderate
- 3 = Average
- 4 = Significant
- 5 = Essential

Add the degrees of influence for all 14 general system characteristics to produce the total degree of influence (TDI)as shown in table III. The UFP is computed using predefined weights for each function type.

iii) Value Adjustment Factor (VAF): The value adjustment factor indicates the general functionality provided to the user of the application. Insert the TDI into the following equation to produce the value adjustment factor.

VAF= (TDI * 0.01) + 0.65

For example, if there are three degrees of influence for each of the 14 GSC descriptions , then TDI= (3*14) =42

VAF = (42 * 0.01) + 0.65

VAF = 1.07

iv) Adjusted FP Count: The adjusted function point count is calculated as,

FP =UFP * VAF.

FP can be used to measure errors/FP, defects/FP, pages of documentation/FP, cost/FP.

TABLE III
WEIGHT VALUE BY FUNCTIONAL COMPLEXITY.

Sr.No	General System Characteristics (GSC)	Degree of Influence(0-5)
1	Data Communications	—
2	Distributed Data Processing	—
3	Performance	—
4	Heavily used Configuration	—
5	Transaction Rate	—
6	Online Data Entry	—
7	End User Efficiency	—
8	Online Update	—
9	Complex Processing	—
10	Reusability	—
11	Installation Ease	—
12	Operational Ease	—
13	Multiple Sites	—
14	Facilitate Change	—
	Total Degree of Influence (TDI)	——

C. Object Point

It measure the size from a different dimension. This mea- surement is based on the number and complexity of the following objects:screens, reports and 3GL components [23], [24]. This is a relatively new measurement and it has not been very popular. But because it is easy to use at the early phase of the development cycle and also measures software size reasonably well. This measurement has been used in COCOMO II for cost estimation. Following are the steps for calculating object point:

i) Assess object count: number of screens, reports and 3GL components.
ii) Classify object: simple, medium and difficult depending on the values of characteristic dimensions.
iii) Weight the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level as given in the following table.

TABLE IV
WEIGHT VALUE BY OBJECT COMPLEXITY.

Object Type	simple	medium	difficult
Screen	1	2	3
Reports	2	5	8
3 GL components			10

iv) Determine object points: add all the weighted object instances to get one number, the object point count.

v) Estimate percentage of reuse you expect to be achieved in this project. Compute new object points to be devel- oped as,

NOP = (ObjectP oint) ∗(100 − %reuse)/100

where, %reuse is the percentage of screens, reports, and 3GL modules reused from previous applications.

vi) Determine a productivity rate depending on developers' experience and ICASE maturity as given in the table,

TABLE V
PRODUCTIVITY.

	Low	Lowest	High	Highest
PROD	4	7	25	50

vii) Compute the estimated person-months as , Person Month = NOP/PROD.

D. Test Point (TP)

Test point used for test point analysis (TPA), to estimate test effort for system and acceptance tests [15]. However, it is important to note that TPA itself only covers black-box testing. Hence, it is always used with FPA, that does not cover system and acceptance tests. Consequently, FPA and TPA merged together provide means for estimating both, white and blackbox testing efforts. There are a lot of dependent and independent factors that need to be taken into account. Since TPA is based upon the principles of a black-box test, requires knowledge of following three basic elements:

i) Size of the software system :

Using TPA, the size of a software system is approxi- mately equivalent to the amount of FP previously calcu- lated through FPA. Nevertheless, again certain factors need to be taken into account, that have alternating influences on the testing size , as opposed to the amount of FP. These factors are complexity, interfacing and uniformity.

ii) Test strategy:

It specifies which quality characteristics are going to be tested for each function, as well as the degree of cover- age. Hereby, more important functions and characteris- tics are going to be tested more intensively. Therefore, it is important to note, that to determine the importance of these characteristics in conjunction with the client to achieve a maximum level of testing conformance, further differentiates between two factors that influence the thoroughness of testing in accordance with the client:

1. User-importance 2. User-intensity Whereas the factor user-importance denotes the level of significance of a particular function, the userintensity describes its level of usage. Thus, whereas a function that is going to be used throughout the day is characterized as more important by the user, a function that is used very rarely will not be marked a high priority. The same principle can be applied to the user-intensity factor.

iii) Level of Productivity:

It is defined as the amount of time needed to realize a single test point, as determined by the test strategy and the systems size. Hereby, productivity itself is composed of two components, the productivity and the environ- mental factor. The productivity factor is specific to the individual organization, represented by the knowledge and skill of the test team. environmental factor describes the external influence of the environment on the test activities including the availability of test tools, the level of the teams experience with the test environment, the test basis quality as well as the availability of testware.

Thus, the overall procedure behind TPA, is based upon the following steps

i) Dynamic Test Points : dynamic test points calculated, as the sum of the TP assigned to all functions. Hereby, TP are calculated for each individual function using the amount of FP, function dependent factors (user- importance, user-intensity, complexity, uniformity and interfacing), as well as quality requirements.

ii) Static Test Points : are a result of determining the number of test points required to test static measurable characteristics.

iii) Total Amount of TP : is the sum of dynamic and static TP.

iv) Primary Test Hours: represent the volume of work required for the primary testing activities like prepara- tion, specification, execution and completion test phases. Primary test hours can be calculated using the following formula:

$$\mathbf{P}(\mathrm{TP}) * \mathrm{environmentalfactor} * \mathrm{productivityfactor}.$$

v) Total Estimated Test Hours : it is calculated by consid- ering control factor, that represents the volume of work needed for additional management activities. Conversion factor depends on the availability of management tools, as well as the size of the test team.

E. Use Case Point (UCP)

Test effort estimation using UCP [15], [17], [24] is based upon use cases (UC). UC is a systems behavior under various conditions, based on requests from a stakeholder. UC capture contractual agreements between these stakeholders about the systems behavior. Thus, the primary task of UCP is to map use cases (UC) to test cases (TC). Hereby, each scenario together with the corresponding exception flow for each UC serves as input for a specific TC. Basically, the amount of test cases identified through this mapping results in the corresponding test effort estimation. UCP is comprised of six basic steps that determine a projects required test effort:

i) Calculate Unadjusted Actor Weights (UAW)

It is the sum of all actors multiplied by corresponding actor weights, based on the actor type, as shown in table VI, using

following formula
UAW = Actor Weight * Actortype.

TABLE VI
UCP ACTOR WEIGHTS (AW).

Actor Type	Description	Factor
simple	GUI	1
average	Interactive or protocol driver interface	2
complex	API/ low level interactions	3

ii) Determine Unadjusted UC Weights (UUCW)
The amount of UC need to be identified in order to de- termine the corresponding UUCW, which is represented as the sum of all UC multiplied by a weight factor depending on the number of transactions or scenarios, as shown in table VII, using the following formula:

$$UUCW = \sum UC * UCW$$

TABLE VII
UCP USE CASE WEIGHTS (UCW).

Actor Type	Description	Factor
simple	¡=3	1
average	4-7	2
complex	¿7	3

iii) Compute Unadjusted UC Points (UUCP)
Based on the UAW and UUCW, the UUCP can be calculated as,
UUCP = UAW +UUCW
iv) Determine Technical and Environmental Factors
Based on the technical complexity factor (TCF) listed in table VIII, called technical and environmental factors (TEF) can be determined, as the sum of the products of weights and assigned values as,

$$TEF = \sum W * AV$$

TABLE VIII
UCP TECHNICAL COMPLEXITY RATINGS..

Factor	Description	Value
T1	Test Tools	5
T2	Documented Inputs	5
T3	Development Environments	2
T4	Test Environments	3
T5	Test Ware Reuse	3
T6	Distributed System	4
T7	Performance Objectives	2
T8	Security Features	4
T9	Complex Interfacing	5

v) Calculate Adjusted UCP (AUCP) AUCP is calculated using following formula

AUCP = UUCP * (0:65 * (0:01 *TEF)

vi) Compute Final Test effort Test effort using UCP can be calculated as a multiplica- tion of the AUCP with a conversion factor: Effort = AUCP * Conversion Factor
the conversion factor represents the test effort required for a language/technology combination, such as plan- ning, writing and executing tests for a single UCP using Enterprise JavaBeans.

III. ESTIMATION TECHNIQUES

Software cost estimation is the process of predicting the amount effort required to build a software system. Software cost estimation is a continuous activity which can start at the first stage of the software life cycle and continues through the lift time. There are various techniques used in software cost estimation [2], [3], [12], [17], [19]. By following Boehm's classification system, these methods are summarize into three categories [3]: expert judgement, algorithmic estimation, and analogy based estimation. In this section some popular esti- mation methods are discussed.

A. Algorithmic methods

The algorithmic methods are based on mathematical models that produce cost estimate as a function of a number of variables, which are considered to be the major cost factors. Any algorithmic model has the form:

Effort = $f(x_1, x_2, , x_n)$

where $x_1, x_2, , x_n$ denote the cost factors i.e. software metrics.

The existing algorithmic methods differ in two aspects: the selection of cost factors, and the form of the function f. These models work based on the especial algorithm. For years, many models have been developed, some of which are described below [23], [17].

1) COCOMO (Constructive Cost Models): This family of models proposed by Barry Boehm (Boehm, 1981), is the most popular method which is categorized in algorithmic methods. This method uses some equations and parameters, which have been derived from previous experiences about software projects for estimation. The models have been widely accepted in practice. In the COCOMOs, the code-size S is given in thousand LOC (KLOC) and Effort is in person-month. Three modles of COCOMO given by Barry Boehm

Simple COCOMO : It was the first model suggested by Barry Boehm, which follows following formula: Effort = $a * (K\,LOC)^b$

where S is the code-size, and a, b are are complexity factors. This model uses three sets of a, b depending on the complexity of the software only as given in table IX. The basic COCOMO model is simple and easy to use. As many cost factors are not considered, it can only be used as a rough estimate.

Intermediate COCOMO: In the intermediate COCOMO, a nominal effort estimation is obtained using the power function with three sets of a, b, with coefficient a being slightly different from that of the basic COCOMO as shown in table X. Previous model

TABLE IX
COMPLEXITY FACTORS FOR SIMPLE COCOMO.

Model	a	b
Organic (Simple in terms of size and complexity)	2.4	1.05
Semi-ditched (Average in terms of size and complexity)	3.0	1.15
Embedded (Complex)	3.6	1.20

doesnot include the factors which can affect the efforts. Effort = $a * (K\,LOC)^b * EAF$

EAF is the effort adjustment factor used in intermediate COCOMO. The overall impact EAF is obtained as the product of all individual factors. Typical values for EAF range from 0.9 to 1.4. The detailed COCOMO works on each sub-system separately and has an obvious advantage for large systems that contain non-homogeneous subsystems.

COCOMO II : Number of effort adjustment factor are increases by 5, now it becomes 22 as shown in table XII.

TABLE XI
EFFORT ADJUSTMENT FACTORS USED IN INTERMEDIATE COCOMO.

Cost Driver	Description
DATA	Database size
CPLX	Product complexity
TIME	Execution time constraint
STOR	Main storage constraint
RUSE	Required reusability
DOCU	Documentation match to life cycle needs
PVOL	Platform volatility
SCED	Scheduling Factor
RELY	Required reliability
TOOL	Use of software tools
APEX	Application Experience
ACAP	Analyst capability
PCAP	Programmer Capability
PLEX	Platform experience
LTEX	Language and tools experience
PCON	Personnel continuity
SITE	Multisite development

TABLE XII
EFFORT ADJUSTMENT FACTORS USED IN INTERMEDIATE COCOMO OTHER THAN INTERMEDIATE COCOMO.

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

TABLE X
COMPLEXITY FACTORS FOR INTERMEDIATE COCOMC

Model	a	b
Organic (Simple in terms of size and complexity)	3.2	1.05
Semi-ditched (Average in terms of size and complexity)	3.0	1.15
Embedded (Complex)	2.8	1.20

2) Puntam's model and SLIM: Putnam's model is proposed according to manpower distribution and examination of many software projects. Software equation for puntam's model is as follows:

$$S = E * Effort^{1/3} * t_d^{4/3}$$

where t_d is the software delivery time; E is the environment factor that reflects the development capability, which can be derived from historical data using the software equation. The size S is in LOC and the Effort is in person-year. Another important relation found by Putnam is

$$Effort = D_0 * t_d^3$$

where D0, is a manpower build-up factor, which ranges from 8 (new software) to 27 (rebuilt software). Com- bining above 2 equations, final equation is obtained as :

$$Effort = D_0^{4/7} * E^{-9/7} * S^{9/7} \text{ and}$$

$$t_d = D_0^{-1/7} * E^{-3/7} * S^{3/7}$$

Putnam's model is also widely used in practice and SLIM is a software tool based on this model for cost estimation and manpower scheduling.

B. Expertise Based Estimation is the most frequently applied estimation strategy for oft- ware projects, that there is no substantial evidence in favor of use of estimation models, and that there are situations where we can expect expert estimates to be more accurate than formal estimation models. This method is usually used when there is limitation in finding data and gathering requirements. Consultation is the basic issue in this method. The following twelve expert estimation best practice guidelines are consid- ered aiming at reducing the size of situational and human biases in expert estimation [21].

- Evaluate estimation accuracy,but avoid high evaluation pressure.
- Avoid conflicting estimation goals.
- Ask the estimators to justify and criticize their estimates.
- Avoid irrelevant and unreliable estimation information.
- Use documented data from previous development tasks.
- Find estimation experts with relevant domain background and good estimation records.
- Estimate top-down and bottom-up, independently of each other.
- Use estimation checklists.
- Combine estimates from different experts and estimation strategies.
- Assess the uncertainty of the estimate.

- Provide feedback on estimation accuracy and develop- ment task relations.
- Provide estimation training opportunities.

Estimation based on Expert judgment is done by getting ad- vices from experts who have extensive experiences in similar projects. Examples of expertised based techniques include Delphi technique [22], [24], Rule Based Systems [24].

1) Delphi : The aim of Delphi method is to combine expert opinion and prevent bias due to positions, status or dominant personalities. Delphi arranges an especial meeting among the project experts and tries to achieve the true information. Delphi includes some steps as,

a) The coordinator gives an estimation form to each expert.
b) Each expert presents his own estimation (without discussing with others).
c) The coordinator gathers all forms and sums up them (including mean or median) on a form and ask experts to start another iteration.
d) steps (b-c) are repeated until an approval is gained.

2) Rule Based Systems: Uses human expert knowledge to solve real-world problems that normally would re- quire human intelligence. Expert knowledge is often represented in the form of rules or as data within the Personnel Capability = Low THEN Risk Level = High

C. Learning Oriented Techniques: It use prior and current knowledge to develop a software estimation model. Neural network and analogy estimation are examples of learning oriented techniques.

1) Neural Networks: are based based on the principle of learning from example. Neural networks are char- acterized in terms of three entities, the neurons, the interconnection structure and learning algorithm.Most of the software models developed using neural networks use backpropagation trained feed forward net- works [24]. The network is trained with a series of inputs and the correct output from the training data so as to minimize the prediction error. Once the training is complete and the appropriate weights for the network arcs have been determined , new inputs can be presented to the network to predict the corresponding estimate of the response variable.
2) Analogy Based Estimation (ABE): is a valid alternative to expert judgement and algorithmic cost estimation [3], [12], [22]. ABE is widely accepted since it derives the solutions by simply imitating the human's problem solving approach. The basic idea of ABE is simple: when provided a new project for estimation, compare it with historical projects to obtain the nearest project to estimate development cost of the new project. Similarity function is the core part of ABE, which measures the level of similarity between two different projects.

The ABE system procedure normally consists of the following four stages :

a) Collect the past projects information and prepare the historical data set.
b) Select current projects features such as Function Points (FP) and Lines of Source Code (LOC), which are also collected with past projects.
c) Retrieval the past projects, estimate the similarities between new project and the past projects, and find the most similar analogues. The commonly used similarity function is the weighted Euclidean Distance:

$$D(P, P') = \sqrt{\sum_{i=1}^{l} wi(f_i - f_i')^2}$$

computer. Depending upon the problem requirement, these rules and data can be recalled to solve problems.

An expert system is built based on IF-THEN rules for representing specialist knowledge gained from a human expert. In this case it is the knowledge about how to estimate project cost. Expert system applies this knowl- edge to make decisions. If the knowledge area is specific enough and well isolated, a reliable cost models based on historical data can be constructed. As a result, the estimation procedure can be automated and the project managers gain the ability to easily and quickly predict the cost. One of the example of rule from a rule based system developed by Madachy is as:

IF Required Software Reliability = Very High AND

where p and p' denote any two members of

the project data set, $\mathbf{f_i}$ and $\mathbf{f_i}$ denote the features of project, l is the number of features in each project, and $\mathbf{w_i}$ is the weight of each feature.

d) Predict the cost of the new project from the chosen analogues by using the solution function. Usually the average of analogues cost is used as solution function.

D. Price-to-win

The software cost is estimated to be the best price to win the project. The estimation is based on the customer's budget instead of the software functionality. For example, if a reasonable estimation for a project costs 100 person-months but the customer can only afford 60 person-months, it is common that the estimator is asked to modify the estimation to fit 60 personmonths effort in order to win the project. This is again not a good practice since it is very likely to cause a bad delay of delivery or force the development team to work overtime [23].

E. Bottom-up

In this approach, each component of the software system is separately estimated and the results aggregated to produce an estimate for the overall system. The requirement for this approach is that an initial design must be in place

that indicates how the system is decomposed into different components [23].

F. Top-down

This approach is the opposite of the bottom-up method. An overall cost estimate for the system is derived from global properties, using either algorithmic or non-algorithmic methods. The total cost can then be split up among the various components. This approach is more suitable for cost estimation at the early stage [23].

IV. RECENT TRENDS

The research of new and more accurate size and effort estimation methods led to revising former models and approaches to this problem. New estimation methods include many variants of Puntam's model, function point analysis, application of fuzzy logic, neural network, test execution effort, and bayesian belief network. This section describes some of the recent trends for software project cost estimation.

A. Bayesian Belief Network

Software cost and effort estimation is the process of fore- casting the software effort to estimate software costs of both development and maintenance. In the past decades, various kinds of software cost and effort estimation methods have been proposed. However, there is no optimal approach to accurately predict the effort needed for developing a software system. Because, the information gathered at the early stages of software system development is insufficient for providing a precise effort prediction. It is a complex activity that requires knowledge of a number of key attributes. Bundle of data is needed, which is often impossible to get in needed quantities. Hence, Bayesian Belief Networks are effective for cost and effort estimation [9], [14]. BBNs are especially useful when the information about the past and/or the current situation is vague, incomplete, conflicting, and uncertain. They are a very effective method of modeling uncertain situations that depend on cause and effect. They are compact networks of probabilities that capture the probabilistic relationship between variables, as well as historical information about their relation- ships. BBNs are very effective for modeling situations where some information is already known and incoming data is uncertain or partially unavailable. An important fact to realize about Bayesian Belief Networks is that they are not dependent on knowing exact historical information or current evidence.

B. Test Execution Effort

Testing is an important activity to ensure software quality. Software testing is becoming more and more important as it is a widely used activity to ensure software quality. Testing is now an essential phase in software development life cycle. Test execution becomes an activity in the critical path of project development. Software quality becomes more and more important in current competitive markets. Testing is a widely used quality assurance activity. Testing activities make up 40% of the total software development effort. In order to estimate the effort needed to design, implement and execute tests special methods are needed. Here we will discuss some related models and techniques [6], [7], [15].

1) Function Point Analysis : is a model-based estimation method. It measures the size of a system by calculating the complexity of system functionalities and counting the function points. The count is then used for estimating the effort to develop it. Capers Jones proposed a testing estimation model based on FPA. In his method, number of test cases can be determined by function points as, $\mathrm{NumberOfTestCases} = \mathrm{FunctionPoints}^{1.2}$ Then the effort in personal-hours is calculated with a conversion factor obtained from historical data.

2) Use Case Point Analysis: is an extension of FPA and estimates system sizes based on use case specifications. Suresh Nageswaran [25] presented test effort estimation using UCP. In the method, technical complexity factors for testing are proposed and calculated as the technical and environmental factors to adjust Unadjusted Use Case Point (UUCP). Then a conversion factor is introduced to estimate testing efforts by multiplying the Adjusted Use Case Point (AUCP). UCP-based methods take testing activities including test planning, test design, test execution and test reporting as a whole, but it doesnt give estimation for each individual test unit such as a test case or a test suite.

In 2007, Eduardo Aranha and Paulo Borba [6] presented an estimation model for test execution effort based on formal test specifications. In the model, they defined and validated a measure of size and execution complexity of test cases. This measure is obtained from test specifi- cations written in a controlled natural language. Every step in every test case is analyzed based on a list of characteristics to obtain its execution points. Similar to FP-based and UCP-Based methods, test execution effort is estimated via multiplying execution points with a conversion factor.

V. CONCLUSION

This paper focus on the existing software estimation methods. Also, presented background information on software project models and software metrics to be used for effort and cost estimation. No model can estimate the cost of software with high degree of accuracy. Estimation is a complex activity that requires knowledge of a number of key attributes. At the initial stage of a project, there is high uncertainty about these project attributes. As we learn that BBNs are especially useful when the information about the past and/or the current situation is vague, incomplete, conflicting, and uncertain. Conventional estimation techniques focus only on the actual development effort furthermore, this paper also described test effort estimation. In fact, testing activities make up 40% total software development effort. Hence, test effort estimation is crucial part of estimation pro- cess.

REFERENCES

[1] Jin Yongqin, Li Jun, Lin Jianming, Chen Qingzhang, "Software Project Cost Estimation Based On Groupware", World Congress on Software Engineering, IEEE, 2009.

[2] Chen Qingzhang, Fang Shuojin, Wang Wenfu, "Development of the Decision Support System for Software Project Cost Estimation", World Congress on Software Engineering,IEEE, 2009.

[3] Y. F. Li, M. Xie, T. N. Goh, "A Study of Genetic Algorithm for Project Selection for Analogy Based Software Cost Estimation, IEEE, 2007.

[4] Yinhuan Zheng, Yilong Zheng, Beizhan Wang, Liang Shi, "Esti- mation of software projects effort based on function point", 4th International Conference on Computer Science and Education, 2009.

[5] Pichai Jodpimai, Peraphon Sophatsathit, and Chidchanok Lursin- sap, "Analysis of Effort Estimation based on Software Project Models", IEEE, 2009.

[6] Eduardo Aranha, Paulo Borba, "An Estimation Model for Test Execution Effort", First International Symposium on Empirical Software Engineering and Measurement, IEEE, 2007.

[7] Xiaochun Zhu, Bo Zhou, Li Hou, Junbo Chen, Lu Chen, "An Experience-Based Approach for Test Execution Effort Es- timation", 9th International Conference for Young Computer Scientists, IEEE, 2008.

[8] Hao Wang, Fei Peng, Chao Zhang, Andrej Pietschker, "Software Project Level Estimation Model Framework based on Bayesian Belief Networks", Sixth International Conference on Quality Software (QSIC'06), IEEE, 2006.

[9] angyang Yu, Charlottesville, "A BBN Approach to Certifying the Reliability of COTS Software Systems", annual reliability and maintainability symposium, IEEE, 2003.

[10] Ying Wang, Michael Smith, "Release Date Prediction for Telecommunication Software Using Bayesian Belief Networks", E Canadian Conference on Electrical and Computer Engineer- ing, IEEE, 2002.

[11] Khaled Hamdan, Hazem El Khatib, Khaled Shuaib," Practical Software Project Total Cost Estimation Methods", MCIT 10, IEEE, 2010.

[12] Jairus Hihn, Hamid Habib-agahi, "Cost Estimation of Software Intensive Projects:A Survey of Current Practices", IEEE, 2011.

[13] Khaled Hamdan, Mohamed Madi," Software Project Effort: Different Methods of Estimation", International Conference on Communications and Information Technology (ICCIT), Aqaba.,IEEE,2011.

[14] S. Bibi, I. Stamelos, L. Angelis, "Bayesian Belief Networks as a Software Productivity Estimation Tool,IEEE.

[15] Matthias Kerstner, " Software Test Effort estimation Methods", 2 February 2011.

[16] Nancy Merlo Schett, "Seminar on software cost estimation", University of Zurich,Switzerland, 2003.

[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] Yunsik Ahn, Jungseok Suh, Seungryeol Kim, Hyunsoo Kim, "The software maintenance project effort estimation model based on function points", Journal of software maintenance and evoloution, 2003.

[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] Pressman, Roger S., "Software Engineering: A Practioner's Approach", 6th Edn., McGraw-Hill New York, USA., ISBN: 13: 9780073019338,2005.

[21] M. JORGENSENA, "Review of Studies on Expert Estimation of Software Development Effort", Journal of Systems and software,70(1-2):37–60, 2004.

[22] Vahid Khatibi, Dayang N. A. Jawawi, "Software Cost Estimation Methods: A Review", Journal of Emerging Trends in Computing and Information Sciences, ISSN 2079-8407, Volume 2 No.1, pg. no 21-29,2010-11.

[23] Hareton Leung, Zhang Fan, "Software Cost Estimation", Article 2001.

[24] Bogdan Stepien, "Software Development Cost Estimation Meth- ods and Research Trends",Computer Science, Vol.5,2003.

[25] Suresh Nageswaran, "Test Effort Estimation Using Use Case Points", Quality Week , San Francisco, California, USA, June 2001

