Concerning Data Mining Technique In Selection Of Attributes To Estimate Effort And Cost

K.Gayathiri [1], Dr. T. Nalini [2],
[1,] *Student, M.TECH, Department of CSE, Bharath University, Chennai, Tamil Nadu, India.*
[2,] *Professor, Department of CSE, Bharath University, Chennai, Tamil Nadu, India.*

Abstract: *Software Cost Estimation can be described as the process of predicting the most realistic effort required to complete a software project. Due to the strong relationship of accurate effort estimations with many crucial project management activities, the research community has been focused on the development and application of a vast variety of methods and models trying to improve the estimation procedure. The rapidly increased need of large-scaled and complex software systems leads managers to settle SEE (software effort estimation) as one of the most vital activities that is closely related with the success or failure of the whole development process. Data mining tool is used for selecting a subset of highly predictive attributes such as project size, development, and environment related attributes, typically a significant increase in estimation accuracy can be obtained.*

Keywords *: Regression, Effort estimation, CART, COCOMO, RiR, OLS, LOOCV, CBR*

I. INTRODUCTION

Most information for initial cost estimation comes from the feasibility study and requirement analysis. when we estimate costs very early, there is less information upon which to base estimates and, therefore, this information is less detailed and the estimates may be less accurate. Software cost estimation is important for making good management decisions. It is also connected to determining how much effort and time a software project requires. Cost estimation has several uses:

- It establishes a firm, reliable budget for an in-house project.
- it facilitate competitive contract bids.
- It determines what is most cost effective for the organization.

Software cost estimation provides the vital link between the general concepts and techniques of economic analysis and the particular world of software engineering.There is no good way to perform a software cost-benefit analysis, break-even analysis, or make-or-buy analysis without some reasonably accurate method of estimating software costs, and their sensitivity to various product,project, and environmental factors.These methods provide an essential part of the foundation for good software management.Without a reasonably accurate cost-estimation capability, software projects often experience the following problems:

1. Software project personnel have no firm basis for telling a manager, customer, or salesperson that their proposed budget and schedule are unrealistic. This leads to optimistic overpromising on software development, low-balling on competitive software contract bids, and the inevitable overruns and performance compromises as a consequence.
2. Software analysts have no firm basis for making realistic hardware-software trade-off analysis during the system design phase. This often leads to a design in which the hardware cost is decreased, at the expense of an even larger increase in the software cost.
3. Project managers have no firm basis for determining how much time and effort each software phase and activity should take.This leaves managers with no way to tell whether or not the software is proceeding according to plan. That basically means that the software portion of the project is out of control from its beginning.

1.1 Estimation Model Development Procedure:

The interests of the organization are best served when it develops its own estimation model. the models that will be presented provide the organization with an initial starting point, but organization-specific factors must be determined. To develop an estimation model we have to follow these steps:

1. Determine a list of potential / most important effort cost drivers.
2. Determine a scaling model for each effort and cost driver.
3. Select initial estimation model.
4. Measure and estimate projects and compare.
5. Evaluate quality of estimation as part of project post-mortem.
6. Update and validate model at appropriate intervals.

1.2. Cost Drivers:

When determining the cost of a project, there are cost drivers which must be taken into consideration. Cost drivers are those variables which influence the actual dollar amount spent on any given project. The cost drivers can vary from project to project. In addition, cost drivers can be influenced by the model employed to estimate the cost. One of the major Cost Drivers in many estimation models is the size of the software product. The size of the software product:

- is one of the major cost drivers in software development projects.
- is also the weakest link in the estimation process.
- Needs extra attention if an organization is developing its own model.

1.3.Size Estimation:

Function Points is one method that has been developed from empirical evidence. Many projects were examined with respect to their different characteristics and the size of the final products examined, and finally a model produced to fit the data. Function Points method work best with business type information processing applications, since those are the projects that were examined to produce the model. It is also not as widely known, used, or trusted for estimating SIZE as COCOMO is for estimating EFFORT. The five items that determines the ultimate complexity of an application.Take the weighted sum of these counts to get the number of function points (**FP**) in a system.

Characteristic	**Weight**
Number of Inputs	4
Number of Outputs	5
Number of Inquiries	4
Number of Files	10
Number of Interfaces	7

Once the number of function points (**FP**) is calculated (from requirements), other data is used to estimate module / system SIZE. For various languages, data was obtained on how many lines of code are usually needed to implement one function point.

1.4 Example:

Suppose that Ada program needs 70 LOC per FP. A system with, 10 inputs, 10 outputs, 5 Inquiries, 4 Files and and 2 Interfaces would have an FP equal to:

FP= (10 * 4) + (10 * 5) + (5 * 4) + (4 * 10) + (2 * 7) = 164

And the estimated Size would be equal to:

Size = FP * 70 = 164 * 70 = 11480 LOC

1.5 Software Cost Modeling Accuracy:

Software cost and effort estimation will never be an exact science. Too many parameters such as technical, environmental, personal or political can affect the ultimate cost of software and the effort required to develop it. It is important to recognize that we can't estimate the cost of producing 100,000 source instructions of software as accurately as we can estimate the cost of producing 100,000 transistor radios. There are many reasons for this; some of the main ones are:

- Source instructions are not a uniform commodity, nor are they the essence of the desired product.
- Software requires the creativity and co-operation of human beings, whose individual and group behaviour is generally hard to predict.
- software has a much smaller base of relevant quantitative historical experience and it is hard to add to the base by performing small controlled experiments.

Today, a software cost estimation model is doing well if it can estimate software development costs within 20% of the actual costs, 70% of the time, within the class of projects to which it is calibrated.Currently the Intermediate and Detailed COCOMO models do approximately this well (within 20% of the actual costs, 68 to 70% of the time) over a fairly wide range of applications.

II. LITERATURE SURVEY

In the field of software effort estimation, the effort required to develop a new software project is estimated by taking the details of the new project into account. The specific project is then compared to a historical data set (i.e., a set of past projects) containing measurements of relevant metrics (e.g., size, language used, and experience of development team) and the associated development effort. The first approaches to estimate software development effort were introduced in the late 1960s [1], and relied on expert judgment. In these cases, a domain expert applies his or her prior experience to come up with an estimation of the needed effort. A number of different variations exist, e.g., Delphi expert estimation, in which several experienced developers formulate an independent estimate and the median of these estimates is used as the final effort estimation [2]. While still widely used in companies, an Expert-driven approach has the disadvantage of **lacking an objective underpinning**. Furthermore, the domain expert is key to the estimation process, inducing additional risks. During the last 30 years, a number of formal models for software effort estimation have been proposed such as Cocomo [3], Cocomo II [4], SLIM [5], and Function Points Analysis [6]. These models have some advantages, providing a formulaic underpinning of software effort estimation. Hence, these models allow for a number of analyses to be performed upon the obtained results. For example, Finnie et al. [7] compared Artificial Neural Networks (ANN) and Case-Based Reasoning to Ordinary Least Squares regression (OLS regression). It was found that both artificial intelligence models (ANN and CBR) outperformed OLS regression and thus can be adequately used for software effort estimation. However, **these results were not statistically tested**.

Briand et al. [8], while performing a comparison between OLS regression, stepwise ANOVA, CART, and CBR, found that case-based learning achieved the **worst results**, while CART performed best; however, the difference was not found to be statistically significant. In a follow-up study using the same techniques on a different data set, different results were obtained, i.e., stepwise ANOVA and OLS regression performed best. Although a large variety of techniques are available, expert-driven estimation methods are still frequently applied in a business setting. Evidence from other domains suggests that both data mining and formal models could provide more accurate estimates than expert driven estimation methods. Often cited strong points of an analytical approach are consistency (provided with the same input, a model will always reach the same conclusion) and the fact that such models posses the ability to correctly assess the impact of different inputs [9].

This conjecture was, however, not confirmed by studies in the domain of software effort prediction. Jørgensen stated that "The use of models, either alone or in combination with expert judgment may be particularly useful when 1) there are situational biases that are believed to lead to a strong bias toward over optimism ; 2) the amount of contextual information possessed by the experts is low; and 3) the models are calibrated to the organization using them." [10].

III. PROBLEM STATEMENT

Without a reasonably accurate effort-estimation capability, software projects often experience the following problems:

- Software project personnel have no firm basis for telling a manager, customer, or salesperson that their proposed budget and schedule are unrealistic. This leads to optimistic overpromising on software development, low-balling on competitive software contract bids, and the inevitable overruns and performance compromises as a consequence.
- Software analysts have no firm basis for making realistic hardware-software trade-off analysis during the system design phase. This often leads to a design in which the hardware cost is decreased, at the expense of an even larger increase in the software cost.

- Project managers have no firm basis for determining how much time and effort each software phase and activity should take. This leaves managers with no way to tell whether or not the software is proceeding according to plan. That basically means that the software portion of the project is out of control from its beginning.

IV. EXPERIMENTAL RESULTS AND OVERVIEW

In this section, we test the implementation efficiency of algorithm and compare with existing techniques. Weka tool is used to select the attributes from dataset. All others are written in java. The datasets for these experiments are from NASA Cocomo dataset.

4.1 DATASET INFORMATION

The main reason for this selection was that these datasets have been extensively used to empirically validate or justify a large amount of research results, whereas they are also publically available. Each dataset contains different number of projects and a set of independent variables with mixed-type characteristics, whereas the dependent variable that has to be predicted is the *actual effort*. Another criterion for the selection of the datasets was the ability to apply all the competitive prediction methods on them. Therefore, we did not consider datasets with too many categorical variables which cause problems to certain methods like regression and Neural Networks.

The data sets typically contain a unique set of attributes that can be categorized as follows:

- Size attributes are attributes that contain information concerning the size of the software project. This information can be provided as Lines Of Code (LOC), Function Points, or some other measure. Size related variables are often considered to be important attributes to estimate effort.
- Environment information contains background information regarding the development team, the company, the project itself (e.g., the number of developers involved and their experience), and the sector of the developing company.
- Project data consist of attributes that relate to the specific purpose of the project and the project type. Also attributes concerning specific project requirements are placed in this category.
- Development related variables contain information about managerial aspects and/or technical aspects of the developed software projects, such as the programming language or type of database system that was used during development.

V. PREPROCESSING

Since some of the techniques are unable to cope with missing data (e.g., OLS regression), an attribute is removed if more than 25 percent of the attribute values are missing. Otherwise, for continuous attributes, median imputation is applied in line. In case of categorical attributes, a missing value flag is created if more than 15 percent of the values are missing; else, the observations associated with the missing value are removed from the data set. Since missing values often occur in the same observations, the number of discarded projects turned out to be low.In case of data sets containing more than 100 observations, hold out splitting is applied; otherwise, leave one out cross validation (LOOCV) is used. In case of LOOCV, iteratively one observation is selected as the test set while the remaining observations are used as the training set. The total error is found by summing the errors on the test set (or taking the average of the errors depending on the evaluation metric) in each step.

ATTRIBUTE INFORMATION

```
@attribute rely numeric
@attribute data numeric
@attribute time numeric
@attribute stor numeric
@attribute pcap numeric
@attribute lexp numeric
@attribute modp numeric
@attribute tool numeric
@attribute loc numeric
@attribute actual numeric
@data
0.84,1.20,1.02,1.07,1.15,1.07,1.04,1.2,223,2245
```

%		very low	low	nominal	high	extra high	productivity high range
%	acap	1.46	1.19	1.00	0.86	0.71	2.06
%	pcap	1.42.	1.17	1.00	0.86	0.70	1.67
%	aexp	1.29	1.13	1.00	0.91	0.82	1.57
%	modp	1.24.	1.10	1.00	0.91	0.82	1.34
%	tool	1.24	1.10	1.00	0.91	0.83	1.49
%	vexp	1.21	1.10	1.00	0.90	1.34	
%	lexp	1.14	1.07	1.00	0.95	1.20	
%	sced	1.23	1.08	1.00	1.04	1.10	e
%	stor		1.00	1.06	1.21	1.56	-1.21
%	data		0.94	1.00	1.08	1.16	-1.23
%	time		1.00	1.11	1.30	1.66	-1.30
%	turn		0.87	1.00	1.07	1.15	-1.32
%	virt		0.87	1.00	1.15	1.30	-1.49
%	cplx	0.70	0.85	1.00	1.15	1.30	1.65 -1.86
%	rely	0.75	0.88	1.00		1.15	1.40 -1.87

%

V. ESTIMATION MODELS

Estimating models have been generated by measuring certain properties and characteristics (duration, cost, team size, disk usage, etc...) of past successful projects.Curves are then fit to the data points to get descriptive equations which are the models. We hope these equations can be used for prediction. The fact that they came from measurements of past projects is what gives the hope.

5.1 Small Projects:

Measurements of small to moderate projects resulted in a model of the following form:

$EFFORT = a * SIZE + b$

Here, the magnitude of the effort is a linear function of the size of the project (Number of Lines of Code, usually). This model holds up until a certain point, usually for projects that can be reasonably accomplished by small teams of two or three people. By increasing the size of the project, the above model becomes less and less accurate an the need for a new model increases.

5.2 Large Projects:

Projects requiring teams of more than three or so people tend to behave in the following way:

$EFFORT = a * SIZE^{b}$

Here, the size of the project is scaled exponentially, therefore as a product increases in size, the effort to produce the product grows more than linearly (for $b >= 1$). It means that if we try to develop a larger product, our productivity (*SIZE / EFFORT*) Decreases. This decrease in productivity on larger projects is called a *diseconomy of scale*. The main reasons why larger software products incur diseconomies of scale are the following:

1. Relatively more product design is required to develop the thorough unit-level specifications required to support the parallel activity of a larger number of programmers.
2. Relatively more effort is required to verify and validate the larger requirements and design specifications.
3. Even with a thoroughly defined specification, programmers on a large project will spend relatively more time communicating and resolving interface issues.
4. Relatively more integration activity is required to put the units together.
5. In general, relatively more extensive testing is required to verify and validate the software product.
6. Relatively more effort will be required to manage the project.

Like many other cost estimating models, the COCOMO model use the same equation for estimating the software development effort.

5.3.The Basic COCOMO Model:

The Basic Model makes its estimates of required effort (measured in Staff-Months **SM**) based primarily on your estimate of the software project's size (as measured in thousands of Delivered Source Instructions **KDSI**):

$SM = a * (KDSI)^{b}$

The Basic model also presents an equation for estimating the development schedule (Time of Develop **TDEV**) of the project in months:

$TDEV = c * (SM)^{d}$

Before we can put these equations to much practical use, we need to resolve a number of significant definitional issues, such as:

- *Which Instructions count as delivered source instructions?*
- *Which Staff-Months are included in the estimate?*
- *Which phases are included in "development" ?*
- *What classes of projects are covered by the estimating equations?*

5.4 Definitions and Assumptions:

Below are some additional definitions and assumptions underlying the use of COCOMO.

1. The primary cost driver is the number of delivered source instructions (**DSI**) developed by the project. DSI is defined such that:

- Only source lines that are *DELIVERED* as part of the product are included.Therefore test drivers and other support software are excluded.
- *SOURCE* codes, created by project staff and processed into machine code by some combination of pre-processors, compilers, and assemblers. Code created by applications generators is excluded.
- *INSTRUCTIONS* are defined as lines of code or card images. Declarations are counted as instructions but Comments are excluded.

2. The COCOMO cost estimates only cover the period between beginning of the design phase and end of the integration and test phase. Costs and schedules of other phases (like requirement phase) are estimated separately.
3. A COCOMO Staff-Month (**SM**) consists of 152 hours of working time. This has to be found to be consistent with practical experience with the average monthly time due to holidays, vacation and sick leaves. COCOMO avoids estimating labour costs in dollars because of the large variations between organizations in what is included in labour costs and because **SM** is a more stable unit than dollars. After estimating the effort in **SM** we convert it into dollar estimate by applying different average dollar per Staff-Month figure for each major phase, to account for inflation and the difference in salary level of the people required for each phase.For example, the senior staff (with high salary)are heavily involved in the requirement and early design phases while junior staff (less salary) are more involved in later phases (detailed design, code and test phases) Therefore the average **SM** is higher for early phases.
4. COCOMO estimates assume that the project will enjoy good management by both the developer and the customer.
5. COCOMO assumes that the requirement specification is not substantially changed after the plans and requirement phase.Any significant modification should be covered by a revised cost estimate.
6. The detailed COCOMO model assumes that the influence of the software cost drivers is phase dependent. Basic COCOMO and Intermediate COCOMO do not, except for distinguishing between development and maintenance.

5.5 The Development Mode:

There are several modes of software development .These different software development modes have cost-estimating relationships which are similar in form, but which yield significantly different cost estimates for software products of the same size.In the COCOMO Model, one of the most important factors contributing to a project's duration and cost is the Development mode. Every project is considered to be developed in one of three modes:

- **Organic Mode.**
- **Semidetached Mode**
- **Embedded Mode**

To estimate the effort and development time, COCOMO use the same equations but with different coefficients (*a, b, c, d* in the effort and schedule equations) for each development mode. Therefore before using the COCOMO model we must be able to recognise the development mode of our project

5.6 Organic Mode:

In the organic mode the project is developed in a familiar, stable environment and the product is similar to previously developed products. The product is relatively small, and require little innovation. Most people connected with the project have extensive experience in working with related systems within the organization and therefore can usefully contribute to the project in its early stages, without generating a great deal of project communication overhead. An organic mode project is relatively relaxed about the way the software meets its requirements and interface specifications. If a situation arises where an exact correspondence of the software product to the original requirements would cause an extensive rework, the project team can generally negotiate a modification of the specifications that can be developed more easily.

The Basic COCOMO Effort and schedule equations for organic mode software projects are:

$$SM = 2.4 * (KDSI)^{1.05}$$

$$TDEV= 2.50 * (SM)^{0.38}$$

5.7.Semidetached Mode:

[1] In this mode project's characteristics are intermediate between Organic and Embedded. "Intermediate" may mean either of two things:

[2] An intermediate level of project characteristics.

[3] A mixture of the organic and embedded mode characteristics.

Therefore in an Semidetached mode project, it is possible that:

- The team members all have an intermediate level of experience with related systems.
- The team has a wide mixture of experienced and inexperienced people.
- The team members have experience related to some aspects of the system under development, but not others.

The size of a Semidetached mode product generally extends up to 300 KDSI.
The Basic COCOMO Effort and schedule equations for organic mode software projects are:

$SM = 3.0 * (KDSI)^{1.12}$

$TDEV= 2.50 * (SM)^{0.35}$

VI. EMBEDDED MODE:

In This Development Mode Project Is Characterized By Tight , Inflexible Constraints And Interface Requirements. The Product Must Operate Within A Strongly Coupled Complex Of Hardware, Software, Regulations, And Operational Procedures. The Embedded-Mode Project Does Not Generally Have The Option Of Negotiating Easier Software Changes And Fixes By Modifying The Requirements And Interface Specifications.The Project Therefore Need More Effort To Accommodate Changes And Fixes. The Embedded Mode Project Is Generally Charting Its Way Through Unknown Territory To A Greater Extent Than The Organic Mode Project. This Lead The Project To Use A Much Smaller Team Of Analyst In The Early Stages, As A Large Number Of People Would Get Swamped In Communication Overhead.Once The Embedded Mode Project Has Completed Its Product Design, Its Best Strategy Is To Bring On A Very Large Team Of Programmers To Perform Detailed Design, Coding And Unit Testing In Parallel.Otherwise The Project Would Take Much Longer To Complete. This Strategy As We Will See Leads To The Higher Peaks In The Personnel Curves Of Embedded-Mode Projects, And To The Greater Amount Of Effort Consumed Compared To An Organic Mode Project Working To The Same Total Development Schedule.
The Basic COCOMO Effort and schedule equations for organic mode software projects are:

$$SM = 3.6 * (KDSI)^{1.20}$$

$$TDEV= 2.50 * (SM)^{0.32}$$

Example:

A large chemical products company, is planning to develop a new computer program to keep track of raw materials. It will be developed by an in-house team of programmers and analysts who have been developing similar programs for several years. An initial study has determined that the size of the program will be roughly 32,000 delivered source instructions.

This project is a good example of an organic-mode software project. Using the Basic COCOMO equations for this development mode we have:
Effort :

SM *= 2.4*(32)*$^{1.05}$ *= 91* ***Staff-Months***

Productivity:

*32000****DSI*** */ 91* ***SM*** **=** *352* ***DSI / SM***

6.1 Duration and Staffing:

Once an estimate is obtained for effort (Staff-Month), A manager must determine how many persons to put on the job. This will ultimately determine the calendar duration of the project. It is very important to note that more staff does not mean proportionately less calendar time. More staff complicate communications and this complexity translates into a project slowdown. The second equation of the COCOMO model use the estimated effort of the project (**SM**) to suggest the optimum calendar duration of the project. For the example above with estimated effort of 91 SM we have:

Schedule:

TDEM **=** *2.5 * (91)*$^{0.38}$ *= 14* ***months***

After estimating the duration of the project the manager can easily determine how many persons in the average must be put on the project:

Average Staffing:

91 ***staff-months*** */ 14* ***months*** **=** *6.5* ***FSP (F****ull Time Equivalent* ***S****oftware* ***P****ersonnel* ***)***

6.2 Phase Distribution of Effort and Schedule for Organic Mode:

After estimating the effort an schedule of a project we need to determine how to distribute them among different phases of the project.This distribution varies as a function of the size of the product.Larger software projects require relatively more time and effort to perform integration and test activities, and are able to compress the programming portion of the program by having larger number of peoples programming components in parallel. Smaller software projects have a more uniform, flat distribution of labour throughout the development cycle, and have relatively more resources devoted to the phases other than integration and test.Table 1 and Table 2 present the percentage distribution of the basic software effort and schedule within the development phases of an organic mode product:

Table 1 - Phase Distribution of Effort: Organic Mode

Phase	*Small (2 KDSI)*	*Intermediate(8 KDSI)*	*Medium(32 KDSI)*	*Large(128 KDSI)*
Plans & Requirements	*6%*	*6%*	*6%*	*6%*
Product Design	*16*	*16*	*16*	*16*
Detailed Design	*26*	*25*	*24*	*23*
Code & Unit Test	*42*	*40*	*38*	*36*
Integration & Test	*16*	*19*	*22*	*25*
Total:	*100*	*100*	*100*	*100*

Table 2 - Phase Distribution of Schedule: Oraganic Mode

Phase	*Small (2 KDSI)*	*Intermediate(8 KDSI)*	*Medium(32 KDSI)*	*Large(128 KDSI)*
Plans & Requirements	*10%*	*11%*	*12%*	*13%*
Product Design	*19*	*19*	*19*	*19*
Detailed Design & Code & Unit Test	*63*	*59*	*55*	*51*
Integration & Test	*18*	*22*	*26*	*30*
Total:	*100*	*100*	*100*	*100*

Example:

Using table 1 and table 2 we can calculate the number of staff needed for programming (Detailed Design & Code and Unit Test) phase of the previous example:

Programming Effort :

(0.62) (91 ***SM*** *)= 56* ***Staff-Months***

Programming Schedule:

(0.55) (14)= 7.7 ***months***

Average Staffing:

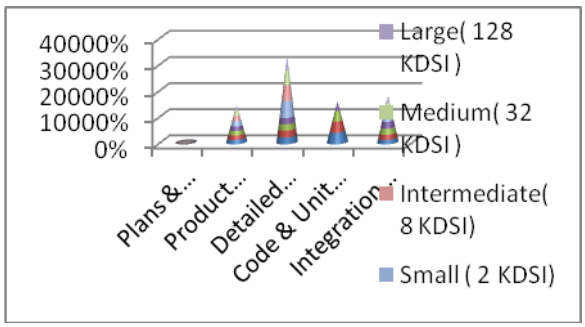
56 ***staff-months*** */ 7.7* ***months*** **=** *7.3* ***FSP (F****ull Time Equivalent* ***S****oftware* ***P****ersonnel* ***)***

6.3 The Basic Labour Distribution:

Using tables 1 and 2, we can calculate the distribution of the effort and schedule among different phases of an organic mode software product. Then by using the same method as the above example we can calculate the number of the personnel required for each phase. Figure 1 shows the basic labour distribution for projects developed in organic mode and with different sizes (Small, Medium, Large). The vertical axis shows

the ratio of the number of the staff in each phase to the average personnel level for whole project. The horizontal axis shows the percent of the scheduled time which has completed.

Figure 1 - Basic Labour Distribution for Softwares Developed in Organic Mode



The shape of the labour curve for larger products have a somewhat higher peak of parallel effort in the programming phase than do the smaller projects.

The COCOMO average labour level estimates for each software development phase yield the straight line, step function labour distribution shown in Fig. 1. Actually, the level of full-time equivalent software personnel active on a project tends more to follow a continuous curve, particularly on large projects, where the instantaneous full-time commitment of a large number of people is an unlikely event. Examples of such curves are also shown in Fig. 1.Rayleigh distribution if properly used yields good approximations to the actual labour curves on many types of software projects.In terms of COCOMO variables we can write the Rayleigh distribution in the following form:

$$FSP = SM\ (t / t_D^2)\ exp(- t / t_D^2)$$

Variable t represents the month for which the FSP (Full-Time Equivalent Software Personnel) is being calculated, and the quantity t_D represents the month at which the project achieves its peak effort. In modeling the full development cycle (from t=0 to TDEV) in the basic COCOMO model, we use the central portion of the Rayleigh Distribution between $0.3t_D$ and $1.7t_D$. therefore we arrive at the labour estimating equation:

$$FSP = SM\ (\ (\ 0.15\ TDEV + 0.7\ t\ ) / 0.25(\ TDEV\ )^2\ )\ exp(\ -\ (\ 0.15\ TDEV + 0.7\ t\ )^2 / 0.5(\ TDEV\ )^2\ )$$

This equation can be used as a reasonably effective means for estimating the personnel level at any given point of an organic-mode software development project.

Phase Distribution of Effort and Schedule for Other Modes:

Table 3 and Table 4 present the percentage distribution of the basic software effort and schedule within the development phases of an semidetached mode product:

Table 3 - Phase Distribution of Effort: Semidetached Mode

Phase	*Small (2 KDSI)*	*Intermediate(8 KDSI)*	*Medium(32 KDSI)*	*Large(128 KDSI)*
Plans & Requirements	*7%*	*7%*	*7%*	*7%*
Product Design	*17*	*17*	*17*	*17*
Detailed Design	*27*	*26*	*25*	*24*
Code & Unit Test	*37*	*35*	*33*	*31*

Integration & Test	*19*	*22*	*25*	*28*
Total:	*100*	*100*	*100*	*100*

Table 4 - Phase Distribution of Schedule: Semidetached Mode

Phase	***Small (2 KDSI)***	***Intermediate(8 KDSI)***	***Medium(32 KDSI)***	***Large(128 KDSI)***
Plans & Requirements	*16%*	*18%*	*20%*	*22%*
Product Design	*24*	*25*	*26*	*27*
Detailed Design & Code & Unit Test	*56*	*52*	*48*	*44*
Integration & Test	*20*	*23*	*26*	*29*
Total:	*100*	*100*	*100*	*100*

Table 5 and Table 6 present the percentage distribution of the basic software effort and schedule within the development phases of an embedded mode product:

Table 5 - Phase Distribution of Effort: Embedded Mode

Phase	***Small (2 KDSI)***	***Intermediate(8 KDSI)***	***Medium(32 KDSI)***	***Large(128 KDSI)***
Plans & Requirements	*8%*	*8%*	*8%*	*8%*
Product Design	*18*	*18*	*18*	*18*
Detailed Design	*28*	*27*	*26*	*25*
Code & Unit Test	*32*	*30*	*28*	*26*
Integration & Test	*22*	*25*	*28*	*31*
Total:	*100*	*100*	*100*	*100*

Table 6 - Phase Distribution of Schedule: Embedded Mode

Phase	***Small (2 KDSI)***	***Intermediate(8 KDSI)***	***Medium(32 KDSI)***	***Large(128 KDSI)***
Plans & Requirements	*24%*	*28%*	*32%*	*36%*
Product Design	*30*	*32*	*34*	*36*
Detailed Design & Code & Unit Test	*48*	*44*	*40*	*36*
Integration & Test	*22*	*24*	*26*	*28*
Total:	*100*	*100*	*100*	*100*

By comparing tables 1 through 6 we can see some differences between the effort and schedule distribution of the products developed in different modes. The main differences are:

1. The embedded-mode project consumes considerably more effort in the integration and test phase. This results from the need to follow and verify software requirements and interface specifications more carefully in the embedded and semidetached mode.
2. The embedded-mode project consumes proportionally less effort in the code and unit test phase. This results from the proportionally higher effort required for the other development phases.
3. The embedded-mode project consumes considerably more schedule in both the plans and requirement phase and the product design phase. This is because of the project's need for more thorough, validated requirements and design specifications, and the greater need to perform these phases with a relatively small number of people.
4. The embedded-mode project consumes considerably less schedule in the programming phase. This results from the strategy of employing a great many people programming in parallel, in order to reduce the project's overall schedule.

6.4. Manpower Distribution Curves and the Rayleigh Distribution:

The Basic COCOMO estimated personnel distribution curves for the medium-size organic, semidetached and embedded mode projects are shown in Fig. 2 , normalized with respect to the average personnel level for each project. The relatively slower, low level start and higher peak personnel requirement of the embedded mode is clearly evident in this figure. These characteristics are even more pronounced for very large embedded-mode projects.This figure also shows the normalized Rayleigh curve :

$$FSP= 13{,}300(t / 60^{2}) exp(- t^{2} / 2(60)^{2})$$

Whose peak effort occurs at the 60% point in the development schedule. Again Rayleigh curve is a reasonably good fit for portions of the manpower distribution with the main exception of its zero-level behaviour at the start of the project. Thus for practical use, we would have to tailor a portion of a Rayleigh curve to a particular mode and a particular portion of the development cycle.In general, it is easier and more realistic to develop a project labour plan from the average personnel per phase information given by the COCOMO model, plus as much as information as you can obtain about the future availability of people to support the project and their needs for advance training, combined with your knowledge of the project's strategy for incremental development

VII . INTERMEDIATE COCOMO MODEL:

The Intermediate COCOMO is an extension of the basic COCOMO model. Here we use the same basic equation for the model. But coefficients are slightly different for the effort equation.Also in addition to the size as the basic cost driver we use 15 more predictor variables. these added cost drivers help to estimate effort and cost with more accuracy. An estimator looks closely at many factors of a project such as amount of external storage required, execution speed constraints, experience of the programmers on the team, experience with the implementation language, use of software tools, etc., for each characteristic, the estimator decide where on the scale of "very low" , " low", " Nominal", "High", "Very High" and "High" the project falls. Each characteristic gives an adjustment factor(from the table 7) and all factors are multiplied together to to give an Effort Adjustment Factor (EAF).If a project is judged normal in some characteristic the adjustment factor will be 1 for that characteristic (Nominal column in Table 7), which means that that factor has no effect on overall EAF. The effort equation for the intermediate model has the form of:

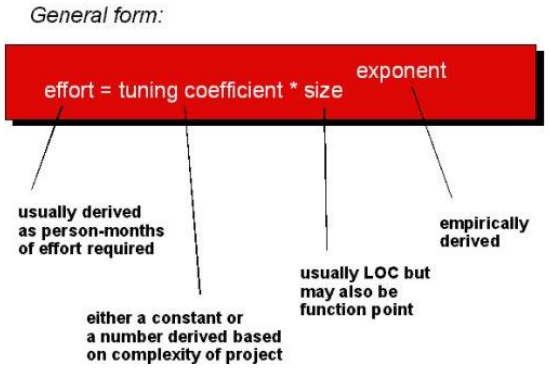
$$SM = EAF * a * (KDSI)^{b}$$

If we assume that the project is "Nominal" in every aspect then all adjustment factors would be 1, which results in EAF=1, and the effort equation would have the same form as the Basic mode.

In addition to the EAF the model parameter "a" is slightly different in Intermediate COCOMO, but the "b" parameter is the same. The effort equation for different modes of Intermediate COCOMO is given in the following table:

Development Mode	Intermediate Effort Equation
Organic	$SM = EAF * 3.2 * (KDSI)^{1.05}$
SemiDetached	$SM = EAF * 3.0* (KDSI)^{1.12}$
Embedded	$SM = EAF * 2.8* (KDSI)^{1.20}$

Figure 2 – Equation for Estimating Effort of Software Developed



Bohem in Software Engineering Economics defines each of the cost drivers, and defines the Effort Multiplier associated with each rating (Table 7).

Table 7 - Project Characteristic Table

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
ACAP Analyst Capability	1.46	1.19	1.00	0.86	0.71	--
AEXP Application Experience	1.29	1.13	1.00	0.91	0.82	--
CPLX Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
DATA Database Size	--	0.94	1.00	1.08	1.16	--
LEXP Language Experience	1.14	1.07	1.00	0.95	--	--
MODP Modem Programming Practices	1.24	1.10	1.00	0.91	0.82	--
PCAP Programmer Capability	1.42	1.17	1.00	0.86	0.70	--
RELY Required Software Reliability	0.75	0.88	1.00	1.15	1.40	--
SCED Required Development Schedule	1.23	1.08	1.00	1.04	1.10	--
STOR Main Storage Constraint	--	--	1.00	1.06	1.21	1.56
TIME Execution Time Constraint	--	--	1.00	1.11	1.30	1.66
TOOL Use of	1.24	1.10	1.00	0.91	0.83	--

Software Tools						
TURN Computer Turnaround Time	--	0.87	1.00	1.07	1.15	--
VEXP Virtual Machine Experience	1.21	1.10	1.00	0.90	--	--
VIRT Virtual Machine Volatility	--	0.87	1.00	1.15	1.30	--

Example:

If your project is rated Very High for Complexity (Effort Multiplier of 1.30), and Low for Tools Use (Effort Multiplier of 1.10), and all of the other cost drivers are rated to be Nominal, these Effort Multipliers are used to calculate the Effort Adjustment Factor, which is used in the effort equation:

EAF = 1.30 * 1.10 = 1.43

Effort = EAF * 3.2 * 31.05 = 14.5 SM

TDEV = 2.5 * 14.50.38 = 6.9 Months

Average staffing:

14.5 / 6.9 = 2.1 people

There are two reasons why Intermediate model produces better results than the Basic Model. First, It considers the effect of more cost drivers. Second, in the Intermediate mode the system can be divided into "components". DSI value and Cost Drivers can be chosen for individual components, instead of for the system as a whole. COCOMO can estimate the staffing, cost, and duration of each of the components--allowing you to experiment with different development strategies, to find the plan that best suits your needs and resources. The net result that we finally hit is Man power and time.

VIII. CONCLUSION

Based upon the background readings, this paper states that the existing models were highly credible; however, this survey found this not to be so based upon the research performed. All the models could not predict the actual against either the calibration data or validation data to any level of accuracy or consistency. Furthermore, it is shown that, typically, a significant performance increase can be expected by constructing software effort estimation models with a limited set of highly predictive attributes by constructing Phase Distribution for Effort and Schedule. Hence, it is advised to focus on data quality rather than collecting as much predictive attributes as possible. Attributes related to the size of a software project, to the development, and to environment characteristics are considered to be the most important types of attributes.

REFERENCES

[1] E.A. Nelson, Management Handbook for the Estimation of Computer Programming Costs. System Developer Corp., 1966.

[2] B. Kitchenham, S. Pfleeger, B. McColl, and S. Eagan, "An Empirical Study of Maintenance and Development Estimation Accuracy," The J. Systems and Software, vol. 64, pp. 57-77, 2002.

[3] B. Boehm, Software Engineering Economics. Prentice Hall, 1981.

[4] B. Boehm, R. Madachy, and B. Steece, Software Cost Estimation with Cocomo II. Prentice Hall, 2000.

[5] L.H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," IEEE Trans. Software Eng., vol. 4, no. 4, pp. 345-361, July 1978.

[6] A.J. Albrecht and J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Trans. Software Eng., vol. 9, no. 6, pp. 639-648 Nov. 1983.

[7] G. Finnie, G. Wittig, and J.-M. Desharnais, "A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models," J. Systems and Software, vol. 39, pp. 281-289, 1997.

[8] L. Briand, K.E. Emam, D. Surmann, and I. Wieczorek, "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques," Proc. 21st Int'l Conf. Software Eng., pp. 313-323, May 1999

[9] R.M. Dawes, D. Faust, and P.E. Meehl, "Clinical versus Actuarial Judgement," Science, vol. 243, no. 4899, pp. 1668-1674, 1989.

[10] M. Jørgensen, "Forecasting of Software Development Work Effort: Evidence on Expert Judgement and Formal Models," Int'l J. Forecasting, vol. 23, pp. 449-462, 2007.

[11]