CHAPTER 4

FAHSCEP: FUZZY AND ANALOGY BASED HYBRID SOFTWARE COST ESTIMATION PROCESS

This chapter elaborates the proposed methodology of Software cost estimation system based on Fuzzy Analogy approach.

4.1 INTRODUCTION

The ultimate objective of software engineering is to design novel methods and devices which are required to build up high-quality applications with added stability and maintainability. With an eye on evaluating and enhancing the quality of an application in the course of the development process, developers and managers invariably employ numerous metrics (Dallal 2010).

The crucial part of most software development process is software cost estimation, it is considered as a major challenging activity. Unfortunately, it is problematic and inaccurate to estimate software development cost. Though there are many estimation approaches and strategies, there is still a need for enhancement.
Estimation models in software engineering are used to calculate some important attributes of future entities—some of them are software development effort, software reliability and productivity. Among these models, the study on estimating software effort has motivated considerable research in recent years. This is due to the growth of software industry in information technology field. Estimation by analogy is one of an attractive technique in software effort estimation field.

4.2 SOFTWARE MEASURES

The efficiency of any software is estimated in accordance with the diverse software parameters such as the reliability, testing coverage, testing time and so on.

4.2.1 Software reliability

The Software Reliability represents the probability of failure free function of software in a specified time duration under given scenarios. The software testing, in essence, is a function to identify the flaws in the configured computer software. In fact, the testing is a very significant means of ensuring the quality of the software by locating various gaps in the software. However, it is significant to note that the testing of the software for a considerably long time does not guarantee a bug-free software and superior consistency. The maximum possible quantity of code has also to be covered to ensure that the software is of superior quality. The testing time by itself may not provide a clear picture of the number of faults rectified in the software. Hence, in order to capture the collective impact of testing time and testing coverage two dimensional software reliability growth models are launched. The excellence of the software is dependent on its consistency to a
very great degree. The error data identification can be realized by assessment of deficits in the graphical representation shown below Figure 4.1.

![Figure 4.1  Data defect detection](image)

In a software product, continuous accessibility is mandatory and hence the consistency plays a vital role in this regard. Normally the consistency of the software is adversely affected because of the deficiencies cropping up during the manufacture of the software. The fundamental reason for the defects in the software is on account of the substandard implementation. The defect is concerned with the determination of the total number of flaws in the software, which are fundamentally generated in the course of the software designing and which are established to be the vital cause for ascertaining the consistency of the software. If the software has to be significantly maintained, the quantity of error in the relative software has to be assessed effectively, so that they can be eradicated altogether. Several innovative models have been launched so as to evaluate the flaw rate cropping up in the software. In this regard, Rayleigh model is the most extensively employed reliability measure model. It is an excellent outfitted model for evaluating the number of deficiencies in the software, in addition
to being the most suitable and simplest model for ascertaining the reliability of the software, being a member of the family of Weibull distribution.

The Rayleigh technique habitually models the deficiency patterns of the whole development process, whereas, the reliability growth model is dependent on the data from the formal testing stages. This goes a long way in ensuring superior reliability test compared to the peer techniques. Generally, this kind of tests are carried out only at the ultimate testing stage. At the time of incidence of breakdown, the deficiencies are defined and well-protected; the software emerges with exceptional stability and added reliability. In this regard, there are two kinds of fundamental models for reliability growth modeling which can be broadly classified using the following features,

- The models well-geared to anticipate the interval between failures
- The models well-equipped to forecast the number of potential failures that may arise in future test intervals (failure count models).

4.2.1.1 Time between failure model

With a view to ascertain the time ($T$) between the breakdowns, the probability density function $F(T)$ is employed, whose parameter estimates are dependent on the value of formerly evaluated breakdown values which is computed as mean time between failures of the system The common expression for anticipated value of time between failure models is expressed by the following Equation.
\[ E[T] = \int_0^\infty T \cdot F(T) dt \] (4.1)

With the help of this equation the reliability of the software over a period of time of length \( l \) arrived, which is furnished by Equation shown below:

\[ \text{Rel}(l) = \int_0^\infty F(T) dt \] (4.2)

By using the above equation, the reliability of the software is effectively worked out.

4.2.1.2 Failure Count Models

In case of reliability models which anticipate well-ahead the number of potential breakdowns in the upcoming test intervals, the probability density function is estimated in accordance with the failure count in the foregoing test intervals. Potential breakdown is the time interval at which the failure occurs frequently.

Let us represent this as \( s \), where \( s \) is a random variable for calculating the failure count, and then let us arrive at the prediction by estimating the expected value of \( s \) as per the below equation.

\[ E[s] = \int_0^\infty s \cdot F(s) ds \] (4.3)
4.2.2 Testing coverage

The Testing coverage is entrusted with the task of discharging one of the most skilled roles in the course of forecasting the software reliability. It can be broadly defined as ‘a structural testing technique in which the software performance can be assessed in relation to specification of the various source codes and the scope or else the degree to which software is implemented by the test cases’. It extends a helping hand to the software designers to estimate the quality of the tested software and arrive at the extent of added activities required to fine-tune the reliability of the software. Further, it boosts the clients with a quantitative confidence benchmark while preparing for the utilization of a software product. Thus, the security of the vibrant system encompasses a high coverage goal. The fundamental testing coverage measures consist of diverse benchmarks like the Statement Coverage, Decision / Condition Coverage, Path Coverage and Function Coverage.

The Statement Coverage, in essence, is defined as the percentage of lines implemented in the distinct program. Let us assume that, the varied flaws are evenly scattered through the length and breadth of the code, then the percentage of executable statements covered illustrates the percentage of flaws uncovered. Figure 4.2 is a general graphical representation for coverage percentage of LOC. This is shown just to notify that effort is directly proportional to coverage % of LOC in testing coverage. The Data is just assumed random values relating to various references gone through. The Decision / Condition Coverage reveals whether Boolean expressions investigated in control structures are assessed as ‘true’ or ‘false’. In General, The Path Coverage gives the percentage of all potential paths available in the code worked out by the test cases. The Function Coverage
gives the diverse percentages of functions/procedures affected by the software testing.

![Graph showing testing coverage](image)

**Figure 4.2  Testing Coverage**

4.2.3  Testing Time

The Testing time indicates the calendar time or the CPU time needed by the software for a process as against the testing effort which involves both the manpower time and the CPU time. During the course of the past twenty years, various methodologies and techniques were designed and spearheaded with prospect of the generation of superior quality, cost-conscious software systems. As a rule, the software development process is home to four different stages such as the requirement phase, design phase, coding and testing. The testing phase is entrusted with the task of identifying and eradicating the developing software flaws with an eye on guaranteeing, to the extent possible, zero-error performance of the software in a specified
time-frame. In short, the testing phase categorizes the quality of the software with respect to its reliability. Thus, the software reliability habitually relies on the number of flaws outstanding in the software.

These are several measures which have to be taken into consideration while designing the specific software. In addition to these measures, cost or effort estimation has also to be deemed as a vital factor. Therefore, in this investigation, effective techniques for software effort estimation in software by means of the soft computing methods are developed. The effort estimation extends a helping hand in comprehending the quality of any software which is required to be developed.

4.3 ESTIMATION PROCEDURE

The estimation models in software engineering are effectively employed to forecast certain significant features of upcoming entities like the software development effort, software consistency and programmer efficiency. Of late, the models evaluating the software effort have urged on elevated enthusiasm of the experiments. In this regard, the estimation by analogy has emerged as one of the noticeable approaches in the domain of the software effort estimation. However, the process employed in estimation by analogy has failed to effectively address the categorical data. The estimation of the work-effort and the agenda needed to design and/or to preserve a software system is one of the most vital functions in the organization of the software projects. This job is labeled as the software cost estimation. In the course of the development task, the cost and time estimates are beneficial for the preliminary uneven authentication and supervision of the project growth. When the task comes to an end, the relative estimates can be used for project productivity evaluation. The
Software cost estimation characterizes the task of forecasting the effort needed to configure a software system. In fact, a major chunk of the cost estimation models produces an effort estimate, which is subsequently transformed into the project duration and cost. Even though the effort and cost are intimately linked, they need not be always associated by a simple transformation function. The effort is generally determined in person months of the programmers, analysts and project managers and is transformed converted into a dollar cost figure by estimating an average salary per unit time of the staff concerned, and thereafter multiplying this by the evaluated effort needed.

4.4 AN EFFICIENT SOFTWARE COST ESTIMATION SYSTEM BASED ON FUZZY ANALOGY

This study presents a proficient process based on reasoning by analogy, fuzzy logic and linguistic quantifiers to estimate effort when the software project is defined either by categorical or numerical data. The Fuzzy logic-based cost estimation models are found to be highly suitable in cases where ambiguous and inaccurate data is to be accounted. They appear to act in the same way as that of the actions of the brain with a rule base. In this new-fangled technique, the effort and the cost are evaluated by means of the Fuzzy logic based analogy technique and the cost can be accurately determined. The efficiency in execution of the proposed system is measured with the help of the MARE (Mean Absolute Relative Error).
4.5 ANALYZE SOFTWARE FUNCTIONAL AND PROGRAMMATIC REQUIREMENTS

The Software estimation involves the scrutiny and filtering of the software functional requirements and the detection of the technical and programmatic parameters and prerequisites. This paves the way for the work elements of the project-specific Work Breakdown Structure (WBS) to become distinctive and the software size and effort to be projected.

For the purpose of assessing and adapting the requisites, the following three steps are initiated:

i) Software functional requirements have to be properly evaluated and adapted to the detailed level. With respect to make the clear risk adjustments, the requirements should be clearly identified that are not well understood. In the domain of the software size estimation, unclear requirements constitute a risk factor which has to be revealed in enhanced uncertainty. If an incremental development approach used, then the sophistication will be according to the requirements that are defined for each and every increment.

ii) Software physical architecture hierarchies according to the operational needs are assessed and modified. In accordance with the software segments to be designed the architecture is defined, and each segment is decayed to minimum feasible level.
iii) In the last and final stage, the project and software plans are assessed to locate the programmatic limitations and requisites embracing the compulsory budgets, schedules, margins, and make or buy conclusions.

4.6 WORK ELEMENTS AND PROCUREMENTS

At this juncture, an endeavour is made to define the work elements and procurements for the software project which has to be integrated into the software estimate. The work element and procurement can be characteristically categorized into the following types of a project-specific WBS:

- Software Management
- Software Systems Engineering
- Software Engineering
- Software Test Engineering
- Software Development Test Bed
- Software Development Environment
- Software System-level Test Support
- Assembly, Test, Launch Operations (ATLO)
- Software Quality Assurance
- Independent Software Verification & Validation (IV&V)

These WBS groups comprise the activities throughout the software life-cycle right from the requirement evaluation study to the ultimate end of the system testing. It is pertinent to note that, the software operations and support (includes the maintenance) are outside the orbit of
these estimates. Work elements like the SQA and IV&V generally are not a part of the software manager’s budget, because quality assurance is something which an organization should deliver and doesn’t come in part of cost, this has been included in most of the existing studies, but, for just the benefit of software managers that these services are being provided by the project. Also SQA is something which an organization certainly needs to deliver; it cannot be quantified in budget.

4.7 ESTIMATE SOFTWARE SIZE

The software size represents the most significant feature which has an effect on the software cost. The objective of this step is to measure the size of the software product. Incidentally, there are essentially two categories of software size metrics - Source Lines of Code (SLOC) and the Function Points. The SLOC, in essence, is an distinctive work of art which calculates the software physical size, though it is habitually employed only during the coding phase, it is very tough to maintain an identical definition across diverse programming languages. The Function Points, in turn, represents a perfect software size metric to evaluate the cost as it is available during the initial development stage like the requirement, determines the software functional size, and continues not to be language independent. The calibration of the Function Points integrates the historical data and offers a further precise picture of the software size.

4.8 EFFORT ESTIMATION

The software effort estimation is an essential trait which directs and assists the planning of software projects. It generally consists of the forecasts of the probable quantity of effort, time, and personnel levels
essential to configure a software system. An incredibly supportive version of effort forecasting is generally made at the childhood phases of a project, when the costing of the project is offered for authorization. As a rule, the effort estimation techniques provide evaluations of the number of work months needed to generate a specified amount of code. Software effort estimation effectively directs the forecast of the probable quantity of effort, time, and personnel levels needed to configure a software system during the early stages of a project. Nevertheless, it is very difficult to get effective evaluations at the initial phases of the project; mainly because of the fact the initial source to evaluate the costing emerges from the requirement specification documents.

4.8.1 Soft Computing Techniques

The Soft Computing techniques are dependent on the data processing in biological systems. The complex biological data processing mechanism empowers the people to stay alive with performing functions such as the detection of the neighborhood, the act of forecasting, planning, and acting accordingly. Human type data processing includes the logical and intuitive data processing. Traditional computer mechanism is well-suited for the logical data processing, but its capacity for intuitive data processing falls well below that of the humans. If a computing system is to have human like data processing skill, it has to be sufficiently adaptable to back the three vital aspects such as the openness, strength, and concurrent processing.

The soft computing varies with the traditional computing also known as hard computing, in such a way that, the former is lenient in regard to inaccuracy, ambiguity, fractional truth, and approximation, whereas the latter is not. In fact, the human mind can be deemed as the role model for
soft computing. The major components such gadgets and methods of Soft Computing (SC) include – the Fuzzy Logic (FL), Neural Networks (NN), Support Vector Machines (SVM), Evolutionary Computation (EC), and Machine Learning (ML) and the Probabilistic Reasoning (PR). The soft computing method is extensively employed in several applications detailed below:

- Application of soft computing to handwriting recognition
- Application of soft computing to automotive systems and manufacturing
- Application of soft computing to image processing and data compression
- Application of soft computing to architecture
- Application of soft computing to decision-support systems
- Application of soft computing to power systems
- Neuro-fuzzy systems
- Fuzzy logic control

The soft Computing characterizes the synthesis of technologies devised to model and facilitate solutions to real-world issues, which are either not modelled or very hard to model, mathematically. It is, in essence, is a consortium of techniques which functions energetically and furnishes, in one way or other, flexible data processing capacity for tackling real-life uncertain scenarios. It is targeted at utilizing the tolerance for inaccuracy, ambiguity, fairly accurate reasoning and fractional truth so as to attain the tractability, strength and cost-effective solutions. The underlying theory is to design the techniques of evaluation which pave the way for a satisfactory solution at minor expenses, by looking for an estimated solution to an inaccurately or accurately generated issue.
4.9 FUZZY LOGIC

The Fuzzy Logic is a unique Soft Computing method essential for assessing complicated mechanisms, particularly where the data configuration is represented by various linguistic constraints. Of late, the Soft Computing methods have surfaced as a realistic option to conventional processes in the assessment and forecast of climate phenomena over the cosmos. This hi-tech mathematical device has thrown a vast prospect for initiative new avenues to thriving atmospheric investigations particularly including risky climate events such as the thunderstorms, hailstorms, excess rainfall, flood and so on. The Soft Computing methods are basically appropriate for the weather assessment and forecast in view of its resistance, inferior data dependency, reduced dependency upon primary conditions, capacity to work upon linguistic data, and the necessary skills to contend with disturbing traits.

The Fuzzy is defined as hazy, blurred, roughly defined, mystified or vague. And the Fuzzy systems represent knowledge-based or rule-based mechanisms. The core of a fuzzy system is deemed as a knowledge base embracing the well-known fuzzy IF-THEN rules. In fact, a fuzzy IF-THEN rule can be considered as an IF-THEN statement, where certain are distinguished by uninterrupted membership functions. As a result the fuzzy logic is capable of being employed to tackle the inaccuracy and ambiguity existing in the initial phases of the project to forecast the effort further precisely by integrating overall lucidity in the forecast mechanism. The Fuzzy logic constitutes a technique, to find solution to the challenges which are highly complicated to be understood quantitatively, in accordance with the fuzzy set hypothesis.
A fuzzy model is generally made use of in cases where the systems are not adequate for assessment by conservative techniques or when the offered data is tentative, erroneous or blurred. The point of Fuzzy logic is to map an input space to an output space by means of a list of if-then statements known as rules. All the rules are evaluated independently, ignoring the order of the rules. For generating the rules, the inputs and outputs of the system are to be recognized. Fuzzy logic flows through the following three phases:

i) Fuzzification
ii) Inference Engine
iii) Defuzzification

The fuzzifier translates the input into linguistic terms by means of the membership functions which indicate the extent to which a specified numerical value of a distinct variable aligns with the linguistic term being taken into account. The fuzzy inference engine carries out the mapping between the input and output membership functions with the help of the fuzzy rules which can be achieved from specialist knowledge of the associations being modeled. A defuzzifier, in turn, performs the Defuzzification function to integrate the output into a single label or numerical value as needed.

4.9.1 Fuzzy membership function

A fuzzy set can be a defined by allocating to each potential individual in the universe of discussion with a value representing its rank of membership in the fuzzy set to a superior or inferior degree as revealed by a smaller or larger membership rating. The input space is also known as the universe of discourse. In this regard, a membership function reveals a curve
which defines the extent to which each point in the input space is mapped to a membership value or degree of membership between 0 and 1. A membership function represents the fuzziness in a fuzzy set and categorizes the element in the set into discrete or continuous. Diverse shapes are employed for graphical representations. Hence, the shape of the membership function plays a vital role. They may have different shapes like triangular, trapezoidal, Gaussian, etc. The only condition a membership Function should really satisfy is that it must vary between 0 and 1. There is a host of several kinds of member functions. The triangular membership function is employed here.

4.9.2 Triangular membership function

It is a three point function defined by a lower limit \( p \), upper limit \( q \) and the modal value in order that \( p < m < q \). The value \( q - m \) is known as margin when it is equivalent to the value \( m - p \). It can be either symmetrical or asymmetrical.

\[
f(x) = \begin{cases} 0 & \text{if } x \leq p \text{ or } x \geq q \\ (x - p)/(m - p) & \text{if } x \in (p, m) \\ (q - x)/(q - m) & \text{if } x \in (m, q) \end{cases}
\] (4.4)

**Fuzziness of Triangular Membership Function (TMF):**

The Fuzziness of a triangular membership function is defined by the below given equation,

\[
fuzziness\text{of TMF} = \frac{\lambda - \mu}{2m}, 0 < TMF < 1
\] (4.5)
Where, $m$ represents the model value, $\mu$ and $\lambda$ indicates the right and left boundaries correspondingly. If the value of fuzziness is greater, the TMF becomes fuzzier.

4.10 EFFORT ESTIMATION BY FUZZY

In Fuzzification, the Triangular Fuzzy number is employed and is defined by:

$$T(S) = \begin{cases} 
0 & \text{if } S \leq a \\
(S-\mu)/(m-\mu) & \text{if } \mu \leq S \leq m \\
(\lambda-S)/(\lambda-m) & \text{if } m \leq S \leq \lambda \\
0 & \text{if } S \geq \lambda 
\end{cases}$$

(4.6)

where, $S$ represents the size as input, $E$ the effort as output, $\mu, m$ and $\lambda$ are the parameters of membership function $T(S)$, $m$ indicates the model value, $\mu$ and $\lambda$ signify the right and left boundaries correspondingly.

Let $(m, 0)$ divide, internally, the base of the triangle in ratio $k:1$, where $k$ is a real positive number. Thus the value of $m$ is furnished by the equation,

$$m = \frac{\mu + k\lambda}{k + 1}$$

(4.7)

Fuzziness is calculated by,

$$F = \frac{\lambda - \mu}{2m}$$

(4.8)
so approximately,

\[ \mu = \left( 1 - \frac{2kF}{k+1} \right)^m \]  

(4.9)

\[ \lambda = \left( 1 + \frac{2F}{k+1} \right)^m \]  

(4.10)

Therefore the TMF \( \delta(E) \) is given as:

\[
\delta(E) = \begin{cases}
0 & \text{if } E \leq a\mu^b \\
\frac{(E/a)^{1/b} - \mu}{m - \frac{\mu}{2}} & \text{if } a\mu^b \leq E \leq am^b \\
\frac{\lambda - (E/a)^{1/b}}{\frac{\lambda - m}{2}} & \text{if } am^b \leq E \leq a\lambda^b \\
0 & \text{if } E \geq a\lambda^b
\end{cases}
\]  

(4.11)

4.11 DEFUZZIFICATION

The output fuzzy estimate of \( E \), is calculated as a weighted average of the optimistic \((a\alpha^b)\), most likely \((am^b)\) and pessimistic estimate \((a\beta^b)\). Fuzzy effort estimate \((E)\) is determined as per the equation 4.12.

\[
E = \frac{w_1(a\alpha^b) + w_2(am^b) + w_3(a\beta^b)}{w_1 + w_2 + w_3} + \sum_{i=1}^{15} EM_i
\]  

(4.12)

where \( w_1, w_2 \) and \( w_3 \) represent the weights of the optimistic, most likely and pessimistic estimate correspondingly and \( EM_i \) specifies the 15 effort
multipliers from COCOMO. The utmost weight has to be given to the most anticipated estimate. For example, if the weight, consider \( w1 \) is the closest weight when compared with threshold, then it is given to the estimate which is to be calculated. The value of \( m \) here indicates the size in KLOC. The values of \( \alpha, \beta, k, F, w_1, w_2 \) and \( w_3 \) represent the arbitrary constants. The effort is achieved in terms of persons per month. Defuzzification has resulted in estimated value based on which the fuzzy rules are being generated.

4.12 FUZZY RULES

The fundamental component of the COCOMO model is employed to design the fuzzy rules to evaluate the nominal effort, irrespective of cost drivers thereby detecting the correspondence between mode, size and the consequent effort by dividing the input and output spaces into fuzzy regions. The constraints of the MFs effort were identified for the specified mode, size pair. 3 MFs characterizing the effort were achieved for a random size and 3 modes correspondingly. The rules configured, according to the fuzzy sets of modes, sizes and efforts as appear in the figure 4.3. The rules are generated according to certain requirement. For example, for calculation of grade based on mark interval, say, above 90 it’s grade ‘O’ and above 80 it’s grade ‘A’. The rules can be applied to any practical scenario where effort has to be calculated based on the conditions.
Figure 4.3  Fuzzy Rules

The current investigation effectively launched an optimized fuzzy logic based framework to tackle the issues of the inaccuracy and ambiguity available in the data in the initial phases of the project to predict the effort more precisely. The captioned structure is constructed upon a modern cost estimation model called COCOMO, which represents an empirical model which is generated by gathering data from a multitude of software projects. These data are assessed to find out the formula that is best-suited to the observations. These formulae connect the size of the system and product, project and team factors to the effort to design the system. In COCOMO, effort is represented as Person Months (PM). The cost drivers possess up to six levels of rating such as the Very Low, Low, Nominal, High, Very High, and Extra High. Each rating possesses a matching real number (effort multiplier), dependent on the factor and the degree with which the factor is capable of affecting an impact on the productivity.
The COCOMO model evaluates the overall effort with respect to the technical project staff. It was designed from the assessment of sixty three (63) software projects. The COCOMO model, in essence, is home to a set of three model such as the Basic, Intermediate, and Detailed. The fundamental COCOMO model estimates the software development effort (and cost) as a function of program size expressed in evaluated lines of code (LOC). The Intermediate COCOMO model evaluates the software development effort as a function of program size and set of cost drivers which encompass the subjective assessments of product, hardware, personnel and project attributes. The comprehensive COCOMO model integrates the entire traits of the intermediary versions with an appraisal of the cost driver's influence on each step such as the analysis and design of the software engineering procedure.

The Product attributes deal with the needed traits of the software product which is in the process of development. The Platform attributes constitute the constraints forced on the software by the hardware platform. The Personnel attributes represent the multipliers which consider the knowledge, skills and competencies of the personnel manning the project. The Project attributes duly take care of the specific traits of the software development project.

The Scale factors (SF) represent the understanding of product goals, flexibility, team coherence, and so on. On the other hand, the Effort multipliers (EF) include the software reliability, database size, reusability, complexity and so on. The ambiguity of the cost drivers has a considerable and adverse effect on the precision of the effort estimates obtained from software effort estimation models. As it is not possible to do away with the ambiguity and indecision of software effort drivers completely, a fuzzy
model sparkles with the qualities of easily authenticating the cost drivers by adopting fuzzy sets.

4.13 COST ESTIMATION

The major motive of this measure is to evaluate the size of the software product. In this regard, there are two important categories of cost estimation techniques such as the algorithmic and non-algorithmic. The algorithmic models differ extensively in mathematical complexity. Certain model is invariably dependent on simple arithmetic formulas employing such summary statistics like the means and standard deviations. Other techniques rely heavily on regression models and differential equations. With an eye on augmenting the precision of algorithmic models, it is highly essential to adapt or calibrate the model to local scenarios. However, these models cannot be employed off-the-shelf. Even with calibration the precision is likely to be quite mixed. The primary measure to estimate cost is to decide the cost of procurements. It involves deciding the cost of support and services like the workstations, test-bed boards and simulators, ground support equipment, and network and communication charges.

This is particularly correct while attempting to align the cost into the budget imposed on the software project. Accordingly, it is highly essential to emphasize the estimates of other steps a number of times, cut down the effort and procurements, or assume additional risk to include into the forced budget. If the schedule becomes widespread, costs are bound to go up as the effort extends to further expensive years. The software costing has to be worked out objectively with the motive of exactly forecasting the expenditure for essential for developing the software. If the project cost has been evaluated as a component of a project bid to the client, a decision has to
be taken regarding the price quoted to the client. Typically, price represents the sum of cost and profit. Figure 4.4 illustrates the overall cost estimation procedure.

![Flow diagram for the entire cost estimation process](image)

**Figure 4.4** Flow diagram for the entire cost estimation process

### 4.14 SUMMARY

The proposed new approach which is based on reasoning by analogy, fuzzy logic and linguistic quantifiers can be used when the software projects are described by categorical and/or numerical data. This approach significantly improves the classical analogy approach which doesn’t consider the categorical data. The categorical data are more frequently available and the processing of the categorical data is very much essential in the modern era as it provides a large number of data required for project evaluation.
Hence analogy approach doesn’t support categorical data processing, fuzzy has become more popular. In the fuzzy analogy approach, both the categorical and numerical data are characterized by fuzzy sets. The advantage of this is to handle correctly the inaccuracy and the uncertainty during the software project. From the implementation results it is observed that the proposed research method effectively estimates the software effort and cost of the software project models.