Software Estimation Techniques

Abdulbasit S. Banga
Assistant Professor, GLS Institute of Computer Technology, GLS Campus, Law Garden, Opp. Ellisbridge, Ahmedabad, State : Gujarat, India.
Mobile: +91 98795 96005, +91 95582 06506
abdulglsmtca@yahoo.co.in and abdulbasit@glsica.org

ABSTRACT
Software cost estimation is the process of predicting the amount of effort required to build a software system. Cost estimates are needed throughout the software life cycle. Preliminary estimates are required to determine the feasibility of a project. Detailed estimates are needed to assist with project planning. The actual effort for individual tasks is compared with estimated and planned values, enabling project managers to reallocate when necessary.

Analysis of historical project data indicates that cost trends can be correlated with certain measurable parameters. This observation has resulted in a wide range of models that can be used to assess, predict, and control software cost on a real-time basis. Models and Techniques provide one or more mathematical algorithms that compute cost as a function of a number of variables.

It is the responsibility of the project manager to make accurate estimates of effort and cost. This is particularly true for projects subject to competitive bidding where a bid too high compared with competitors would result in losing the contract or a bid too low could result in a loss to the organization. This does not mean that internal projects are unimportant. From a project leaders estimate the management often decide whether to proceed with the project. Industry has a need for accurate estimates of effort and size at a very early stage in a project.

The causes of poor and inaccurate estimation:

1. Imprecise and drifting requirements.
2. New software projects are nearly always different from the last.
3. Software practitioners don't collect enough information about past projects.
4. Estimates are forced to match the resources available.

1.1 SOFTWARE SIZING MEASURES:

While estimating is probably the most common use of a sizing measure, there are many other potentially valuable applications, including progress measurement, change management, risk identification, and earned value. There are only two software sizing measures widely used today -- Lines of Code (LOC or KLOC) and Function Points (FP). Though each is a sizing measure, they actually measure different things and have very different characteristics.

Lines of Code is a measure of the size of the system after it is built. It is very dependent on the technology used to build the system, the system design, and how the programs are coded. The major disadvantages of LOC are that systems coded in...
different languages cannot be easily compared and efficient code is penalized by having a smaller size. Capers Jones stated at a talk to the Chicago Quality Assurance Association on November 22, 1996 that anyone using LOC is "committing profession malpractice." Despite these problems, LOC is still frequently used by very reputable and professional organizations. In contrast to LOC, Function Points is a measure of delivered functionality that is relatively independent of the technology used to develop the system. FP is based on sizing the system by counting external components (Inputs, Outputs, External Interfaces, Files and Inquiries). While FP addresses many of the problems inherent in LOC.

1.2 A NEW PARADIGM FROM FUNCTION POINTS OR LINES OF CODE FOR SIZING SOFTWARE SYSTEMS:

Testable Requirements is a new paradigm for measuring of the size of a system. One way to illustrate the differences between lines of code, function points and testable requirements is to look at how each would compare a system developed using a character-based user interface (E.g., DOS, 3270 mainframe), and the equivalent system developed using a graphical user interface (E.g., Windows).

- **LOC perspective:** It would be very hard to make a comparison using LOC because of the differences in the way the interfaces are developed. Since the "language" used to code screen or window functionality is not procedural, it is very difficult to measure LOC, let alone make valid comparisons between two different technical platforms.

- **Function Point Perspective:** Function points would consider them equivalent since they are providing the same functionality to the user (e.g., the same number of logical inputs and outputs). The value adjustment factor could be used to make the GUI a higher function point count. (The value adjustment factor is essentially a complexity factor used to adjust a raw function point count because of factors related to the complexity of the system.)

- **Testable Requirements Perspective:** Testable requirements would give a much higher count to a graphical user interface (GUI) than its character-based equivalent. There are simply more conditions (i.e., testable requirements) that are supported by a GUI than are supported by a character-based interface.

  For example, there is usually only one way to enter a command in a character-based interface - through the menu. In the equivalent GUI, there are usually at least three - the menu, the keyboard, and the icon.

  Testable requirements can also be used to measure and analyze a system in ways that are not possible with other measures. Because testable requirements can measure external user requirements as well as internal technical requirements, it is possible not only to size the user requirements, but also to quantify their impact on the technical design. Performance requirements, for example, may contribute relatively few testable requirements when viewed from an end user perspective. The performance requirements may, however, require a complex real time technical design to meet the requirements and therefore require many testable requirements when viewed from a technical perspective.

  Similarly, some types of systems such as process control and scientific systems may have relatively few external requirements, but have a very complicated internal technical design. The testable requirements paradigm does not require a complexity factor to account for this. Rather the complexity will manifest itself in the size of the design or the size of individual programs. Complexity, in essence, means that there are more testable requirements somewhere.

Major Types of Software Cost Estimates Techniques are following:

1) ALGORITHMIC MODELS
2) EXPERT JUDGEMENT
3) MACHINE LEARNING

2. ALGORITHMIC COST MODELS:

To date most work carried out in the software cost estimation field has focused on algorithmic cost modeling. In this process costs are analyzed using mathematical formulae linking costs or inputs with metrics to produce an estimated output. The formula used in a formal model arise from the analysis of historical data. The accuracy of the model can be improved by calibrating the model to your specific development environment, which basically involves adjusting the weightings of the metrics. There are a variety of different models available, the best known are Boehm's COCOMO [BOEHM-81], Putman's SLIM, and Albrecht's function points [ALBR-83]. On an initial instinct you might expect formal models to be advantageous for their 'off-the-shelf' qualities, but after close observation this is regarded as a disadvantage by cost estimators due to the additional overhead of calibrating the system to the local circumstances. However, the more time spent calibrating a formal model the more accurate the cost estimate should be. A distinct disadvantage of formal models is the inconsistency of estimates, [KEMERER] conducted a study indicating that estimates varied from as much as 85 - 610% between predicated and actual values. Calibration of the model can improve these figures, However, formal models still produce errors of 50-100%.

In terms of the estimation process, nearly all algorithmic models deviate from the classical view of the cost estimation process.

An input requirement of an algorithmic model is to provide a metric to measure the size of the finished system. Typically lines of source code are used, this is obviously not known at the start of the project. SLOC is also very dependant on the programming language and programming environment, this is difficult to determine at an early stage in the problem especially as requirements are likely to be sketchy. Despite this SLOC has been the most widely used size metric in the past, but current trends indicate that it is fast becoming less stable. This is probably due to the changes in software development process.
in recent years highlighted with a tendency to use prototyping, case tools and so forth. An alternative is to use.

Fig.1: Classical view of the algorithmic cost estimation process

Function points proposed by [ALBRECHT], which are related to the functionality of the software rather than its size. A more recent approach is to use object points. This is in comparison a new methodology and has not been publicized in the same depth as function points and SLOC. In essence the method is very similar to function points but counts objects instead of functions. Its recent rise has been prompted by the interest in the object orientation revolution.

Algorithmic models generally provide direct estimates of effort or duration. Effort prediction models take the general form:

\[ \text{effort} = p \times S \frac{1}{\text{productivity rate}} \]

where \( p \) is a productivity constant and \( S \) is the size of the system.

Once the value for \( p \) is known. E.g. productivity = 450 source lines of code per month, making \( p = 0.0022 \) and the size of the system has been estimated at 8500 KLOC.

\[ \text{effort} = 0.0022 \times 8500 \]

\[ \text{effort} = 18.7 \text{ person months} \]

The example above assumes that the relationship between effort and size is a linear one. Most models allow for non-linear relationships by introducing economies or diseconomies of scale. The general formula being:

\[ \text{effort} = p \times S^e \]

A study published by [WALSTON & FELIX] which consisted of 60 projects at IBM federal systems division concluded that effort could be modeled as:

\[ \text{effort} = 5.2 \text{ KLOC}^{0.91} \]

This is an example of economies of scale. As the exponent 0.91 is less than 1. Walston and Felix observed the following results:

<table>
<thead>
<tr>
<th>Effort (PM)</th>
<th>Size (KLOC)</th>
<th>KLOC/PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.27</td>
<td>10</td>
<td>0.24</td>
</tr>
<tr>
<td>79.42</td>
<td>20</td>
<td>0.25</td>
</tr>
<tr>
<td>182.84</td>
<td>50</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Table 1.0

These findings indicate that there is greater productivity when building large software systems as opposed to small systems. However, the results can be justified as it is expected that larger teams can specialize and the overheads are of a relatively fixed size.

3. EXPERT JUDGEMENT:
The majority of research work carried out in the software cost estimation field has been devoted to algorithmic models. However, by an overwhelming majority, expert judgment is the most commonly used estimation method. A Dutch study carried out by [HEMSTRA] revealed that 62% of estimators / organizations use this intuition technique and a study carried out by [VIGDER & KARK] also confirmed the techniques popularity. In its crudest form the expert judgment method involves consultation with one or more local experts who are knowledgeable about the development environment or application domain to estimate the effort required to complete a software project.

The method relies heavily on the experience of their knowledge in similar development environments and historically maintained databases on completed projects and the accuracy of these past projects. However, the study carried out by [VIGDER & KARK] indicated that in general estimators did not refer to previous projects as it was too difficult to access or the expert could not see how the information would help in the accuracy of the estimate. The study claimed that the majority of estimators tended to use their memories of previous projects. If more than one expert is used the weighted average of their estimates are taken. There are obvious risks with this method. As the project may have some unique features which could take longer than anticipated. The weighted average is also very much dependent on the competence of the estimator. However, a particular strength of using an expert is that they can raise unique strengths and weaknesses of the local organizational characteristics.

Despite widespread use, the method seems to have received a rather poor reputation and is often regarded as subjective and unstructured making it vulnerable against more structured Assessment of Expert Judgment:

Advantages:
- Uses experiences on past projects to assess factors on the new project.
- Assessment of representativeness.
- Adapt to exceptional circumstances.

Drawbacks:
- No better than the expertise and the objectivity of the estimator.
- The estimation can be biased.
4. WIDEBAND DELPHI APPROACH:
The Delphi approach or Wideband Delphi technique attempts to gather the opinions of a group of experts with the aim of producing an accurate unbiased estimate. It is a structured technique of expert judgement and is essentially a form based technique involving a multistep procedure:
1. Experts are issued the specification and estimation form by the co-ordinator.
2. A group meeting is held to discuss the product and estimation issues.
3. Experts produce an independent estimate.
4. Estimates are returned indicating the median estimate and the expert’s personal estimate.
5. Another group meeting is held to discuss results.
6. Experts prepare a revised independent estimate.
7. Steps 3-6 are repeated until a consensus is reached by the panel of experts.

4.1 ASSESSMENT OF THE WIDEBAND DELPHI APPROACH:
The method is very good at removing the politics from an estimate, as the experts do not communicate about their particular estimate and the method filters out extreme opinions making the estimate unbiased, which is often at fault in many other estimation techniques. The group discussion is particularly advantageous as it ensures that any estimation issues are not overlooked. The father of all software cost estimation publications produced by [BOEHM 82] indicates the effectiveness of the wideband delphi technique and stresses the importance of the group meeting.

Advantages:
• Removes the politics from an estimate.
• The estimate is not usually biased.
• Group discussion ensures estimation issues are not overlooked.

Drawbacks:
• Can be time consuming, as there are many people involved.
• The panel needs to be very experienced.

5. MACHINE LEARNING:
The continuing poor performance results produced by statistical cost estimation models have flooded the cost estimation area for over the last decade. Their inability to handle categorical data, cope with missing data points, spread of data points and most importantly lack of reasoning capabilities has triggered an increase in the number of studies using non-traditional methods, the latest of which is machine learning techniques. This technique categorizes together the methods:

- Neural Networks
- Fuzzy Logic
- Case-Based Reasoning
- Analogy
- Rule-Based
- Regression Trees
- Hybrid System

Machine learning is a new area which is demonstrating the promise of producing consistently accurate estimates. The system effectively "learns" how to estimate from training set of completed projects. In theory this approach is more robust against noise and data outliers all suggesting machine learning techniques are suitable for software project effort prediction. This is exemplified by a research project undertaken at Bournemouth University by Schofield involving the development of an analogy based case tool called ANGEL.

A neural net is a machine learning technique. Neural networks are organized in layers each consisting of neurons or processing elements that are interconnected. The neurons or perceptrons compute a weighted sum of their inputs, generating an output. If the sum exceeds a certain threshold, for example outputting 1 if the sum is greater than the threshold value or a 0 if otherwise.

As we know it is multi-layered and is known as a feed forward perceptron. The first layer called the input layer contains neurons that represent the set of input variables. The output layer represents the output variable which is the actual effort required to complete the project. The connections between the neurons have weighted numerical inputs associated with them. These weightings are initially random but are adjusted as the perceptrons learn on the training set of past projects.

There a number of learning methods to train neural nets but the back propagation (back-prop) paradigm has emerged as the most popular training mechanism. The back-prop method works by measuring the difference between output and the observed output value. The values being calculated at the output layer are propagated to the previous layers and used for adjusting the connection weights.
Some research studies have reported that neural nets can out perform statistical models [KARUNANITHI] and [WITTIG & FINNIE]. The results produced by the Wittig and Finnie study which used a back-prop multi layer perceptron neural net produced very accurate results but the study indicated that a large training set was required. In contradiction a study performed by Samsom using an Albus multi layer perceptron. Using Boehm's dataset the results compare badly against the Wittig and Finnie results. These inconsistent results lead us to believe that although there is an increasing popularity in this method. There are potential drawbacks which are indicated below:

**Assessment of Neural Nets:**

**Advantages:**
- Results indicate that this approach is comparable, if not better than statistical models.
- Can cope with heterogeneous datasets.
- Many different learning algorithms to choose from**.

**I have identified "Many different learning algorithms to choose from" as an advantage, this is subjective as this point could also be classified as a disadvantage. It may be hard for an estimator to determine which learning algorithm will give the best results for their problem.**

**Drawbacks:**
- No clear guidelines on how to design neural nets, for example how many hidden layers?
- Accuracy of results relies heavily on the size of the training set. Large training sets not always available.
- The logic behind the estimate is hard to convey to the user.
- They are effectively black boxes, once given inputs you have to accept the generated outputs.

6. **FUZZY LOGIC SYSTEMS:**

Fuzzy logic systems have been relatively under used in effort prediction. To date only a few publications have shown to utilize fuzzy logic. A fuzzy system involves mapping a number between 0 and 1 to input variables to indicate the truth value, 1 representing absolute truth. For example we might say Program Y is small and assign it a truth value of 0.7

A fuzzy rule based system can then be constructed, based upon the truth variable above:

- If Program Y small (0.7) THEN development effort = time short (0.8)

The fuzzy rule above indicates that if a truth value of greater than 0.7 then the rule will ‘fire’ and produce a short development time, with a confidence of 0.8.

To make the prediction more accurate many fuzzy rules are required. It is also possible to classify the level of valued logic more closely to the dataset. For example, if we know that the development will be in the region of 10000 LOC then the level of valued logic may have the following:
- quite large
- large
- very large

An example of a fuzzy logic system is presented by [GRAY et al.]. In this example truth values are provided for data model size(30), number of screens(26) and process model size(74). This values are plotted on their respected functions below:

**Assessment of fuzzy systems:**

It is possible for untrained estimators to create intuitive models as the mapping of truth values is relatively easy. However, fuzzy systems suffer from similar symptoms to the rule base machine learning method. It is hard to collect the data on multiple functions. Another weakness identified in the [GRAY] paper indicates that by classifying the levels of valued logic which can make the system more accurate, makes it harder to maintain a degree of meaningfulness as it is harder to...
differentiate between some levels of valued logic (e.g. quite large - large).

Advantages:
• A very intuitive technique, no real formal training required.

Drawbacks:
• Knowledge elicitation stage.
• Hard to maintain a degree of meaningfulness.
• Another use of fuzzy concepts is to combine them to neural net technology to develop hybrid neural-fuzzy systems.

7. CASE BASED REASONING :
Case based reasoning is a method of storing observations on previous projects. Such as effort required to implement the project, programming platform and so forth. When faced with a new observation it is then possible to retrieve the closest stored observations.

7.1 ASSESSMENT OF CBR SYSTEMS:
The performance of CBR systems have shown encouraging results. An experiment conducted by [MUHKOPADHYAY et al.] compared the performance of a CBR system against experts, function points and COCOMO. The performance of the CBR system exceeded both algorithmic models and was extremely close to the level of the expert. As CBR systems use stored cases, it is apparent that as observations are built up the CBR system would have a large knowledge base to compare new observations against. This would suggest CBR systems will get more accurate than experts, as experts have an inability to recall all of their experience and are potentially biased.

Advantages:
• No expert is required.
• The CBR process is more akin to human thinking.
• They can handle failed cases (i.e. those case for which an accurate prediction was not made).

Drawbacks:
• Case data can be hard to gather.
• Predictions are limited to the cases that have been observed.

Case-base reasoning systems are intended to mimic the process of an expert making decisions based on their experience. The CBR technique is therefore very similar to the analogy software cost estimation technique.

8. ANALOGY :
The analogy cost estimation technique involves comparing one or more completed projects in a similar domain as a means of producing new estimates. This can be achieved by analyzing the data collected from these completed projects against the new project, assessing similarities. As the development effort is known this is used as a basis for estimating the current project. Estimation by analogy seems straightforward, but there are a number of problems that need to be addressed, how to determine project features? These are restricted to information that is available at the start of the estimation process, examples of these can include application domain, software process and so forth. Other common problems are how many projects should we measure against? Too few projects can lead to maverick projects being used; too many projects can weaken the effect of the closest analogies. With this estimation method the need for an expert is eliminated, therefore the sometimes difficult knowledge elicitation stage disappears as analogy systems examine problems that have occurred, unlike other cost estimation techniques which need to tackle all possible problems anticipated on a project.

The process involved to predict software cost estimates can be broken down into the following steps according to [COWDEROY & JENKINS]
1) Selection of analogies
2) Assessing similarities and differences
3) Assessment of analogy quality
4) Consideration of any special cases
5) Creating the estimate

8.1 SELECTION OF ANALOGIES:
The first stage in the process is to actually compile a database of suitable completed projects that we can measure similarities against in our new project. The selection process therefore involves selecting analogous projects that reflect both the development environment and project features as the new project.

8.2 ASSESSING SIMILARITIES AND DIFFERENCES:
Now that we have a suitable database of analogous projects, the next stage is to assess the similarities to the new project. Similarity is determined in terms of project features, the number of features depend upon the data available to characterize the projects. [SHEPPERD et al.] reports datasets with as few as one project feature and as many as 29 features being used.

The process of selecting analogies can be achieved by the similarity in n-dimensional space where each dimension corresponds to a project feature. The most similar projects will be those closest together in n-dimensional space. Let's suppose that we want to measure the effort for the new project, by analyzing project features experience, function points and effort. From visualizing 3-dimensional space you can clearly see that project B is closest to the new project and project A would be rejected is this case because of the substantial distance in dimensional space. The example here is rather trivial but in essence it is what happens. A realistic approach would involve more projects and dimensions or project features being used. This allows you to select the closest analogies to the new project, ultimately leading to a more accurate estimate.

8.3 ASSESSMENT OF ANALOGY QUALITY:
To access how well the quality of analogy predications perform you can apply estimation predicate quality measures:
1. Absolute error.
2. Magnitude of relative error (MRE) & mean magnitude of relative error (MMRE).
3. pred (25).
4. Percentage error and mean percentage error.
The results from the quality measures can be fed back into the analogy process to examine the results of why some estimates are better than others.

8.4 CONSIDERATION OF ANY SPECIAL CASES
Sometimes it may be necessary to exclude some projects from consideration. Although the project could have been selected in stage 2 for similarities, that project might have a project feature that you wish to ignore. For example, the project under consideration may have used an unusual design methodology.

Assessment of Analogy:
Advantages:
- No expert is required.
- Analyses problems that actually occur, unlike algorithmic models.
- Adaptive, as you can add new projects.
- Can use categorical data.

Drawbacks:
- Requires accurate details on many past projects.
- Assessing the degree of similarity between projects.

Other Reasons to use Analogy
- Poor results from other methods
- It can handle discontinuous data.
- It's ability to reason.

The whole process of storing analogous projects eliminating redundant features to finding analogies can be automated by a PC based software tool known as ANGEL (ANaloGy softwarE tool) developed by Chris Schofield and is available as a shareware copy. This machine learning tool is the focus of a research project being undertaken at Bournemouth University. For further information on the this tool, please click on the angel link.

9. RULE-BASED SYSTEMS:
Rule-based systems have been used in very few cases of effort prediction. A rule-based system compares facts about the new project that are stored in a knowledge base against a stored set of rules. When a match is made the rule will fire, which can create a chaining effect with one rule enabling another rule to fire. The process continues until there is no more rules to fire and the output is presented to the user.

A typical rule for the rule-base might look like this:
IF module_size > 100 AND interface_size > 5 THEN error_prone_module
IF error_prone_module AND developer_experience < 2 THEN redesign_required

If the first rule fires above then this will enable a new fact the error_prone_module. The new fact can then be used as a premise for the second rule. Thus creating a chaining effect.

Assessment of Rule-Based Systems:
Rule-based systems are at a disadvantage, compared to fuzzy systems. As there is no degree of truth involved. All input variables must be either true or false.

Advantages:
- Simplicity of input variables.

Drawbacks:
- Difficult to derive rules.
- No degree of truth.
- Very few cases of being used for effort prediction.
- Hard to maintain a complex rule-base.

9. REGRESSION TREES:
There have been a number of attempts to use regression trees in effort prediction [SRINIVASON & FISHER]. The results of which have shown that they can out perform proprietary models such as COCOMO, SLIM as so forth. Therefore they warrant a mention and should be taken seriously as an effort prediction method.

A regression tree is created by examining what attributes can classify the data and then an algorithm constructs the tree, splitting nodes where appropriate. The example is very simple as the tree only ever splits into two leaf nodes, this is the most common format as it is easy to comprehend and check for logical errors.

Assessment of Regression Trees:
Advantages:
- Can resist data outliers.
- Can be adjusted for new data.
- Easy to understand.
- Suitable for non-numerical variables.

Drawbacks:
- Sensitive to the algorithm used to construct the tree and tree depth.
- Cannot eliminate a value outside the range supplied in the training data.

10. HYBRID NEURAL FUZZY SYSTEMS:
Another use identified in fuzzy concepts is to combine them with neural networks. The standard neural fuzzy hybrid system is based on the inputs into a neural network being transformed into truth values at the input layer. This is a relatively unexplored area in effort prediction. The theory behind the system would indicate that neural nets would receive more crisp and meaningful inputs thus improving the overall output and quality of neural net predictions. An example of this hybrid system is presented in figure 8.

In figure 8 there are two inputs (data model size and process model size) which are assigned truth values at the first input layer. The output to the second layer represents the membership degrees in each valued logic (small, medium, large) which can lead to the rules layer where the rules represent the input weights. The output from this layer indicates whether the rule ‘fires or not’, determining the
activation level at the output membership layer. Finally, the results are combined to give a single effort predication time.

As with ordinary neural nets the net learns on a training set and once the system has been trained it is possible to remove rules to check for acceptability. This is a very important feature as the network could learn undesired relationships from new data, ultimately forgetting old relationships from old data.

Assessment of hybrid neural nets:
Advantages:
• Potentially more accurate than neural nets.
• Removal of unacceptable rules.
• Reduces noise and input errors.

Drawbacks:
• Relatively unexplored area, no real evidence of success.
• Catastrophic errors if you forget to remove an unacceptable rule.
• Same problems as fuzzy systems.

Suitable for small datasets - Small datasets are very problematic when predicting estimation. This heading assess whether the method is viable when using small datasets.

Reasoning Capability - This assess whether a model has viable reasoning capabilities. For example, in the analogy method you can look back at past projects and question whether a project is really a true analogous project.

Adaptive - This addresses whether the method can accept new data without regenerating the whole estimation process.

<table>
<thead>
<tr>
<th>Machine Learning Method</th>
<th>Explains Output</th>
<th>Suitable For small datasets</th>
<th>Reasoning Capability</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Nets</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Case-Based Reasoning</td>
<td>Yes</td>
<td>Partially</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Analogy</td>
<td>Yes</td>
<td>Partially</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Rule-Based</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>No</td>
</tr>
<tr>
<td>Regression Trees</td>
<td>Yes</td>
<td>Partially</td>
<td>Partially</td>
<td>No</td>
</tr>
<tr>
<td>Hybrid System</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>No</td>
</tr>
</tbody>
</table>

Fig. 9: Comparison of Machine Learning Methods

12. CONCLUSION:
Software Cost Estimation is a vital part of the software development process. Software Cost Estimation continues to be a weak link in software project management. So thus, with the help of these various software sizing and estimation techniques and models, software project estimator can easily estimate the various components of the software, like schedule, cost, effort, human resource etc. and will helps a lot to the project manager to track and control the software at various development stages.

The Paper covers both traditional and state of the art methods identifying advantages and disadvantages of each techniques and models and the underlying aspects in preparing cost estimates.

The Paper endow with comparison of various estimation techniques and models, so thus it will be straightforward and very effectual for the estimator to estimate the software.

13. FUTURE SCOPE:
Various Software Estimation Models are derived from these software sizing and Estimation techniques, so thus with the
help of the models better and accurate estimates can be estimated. Software estimation models will provide the guidelines to other software estimation models and techniques which are involved in this vicinity.

Analyzed models will be used to represent the relationship between effort and a primary cost factor such as size.

Defined Cost drivers will be used to adjust the preliminary estimate provided by the primary cost factor.

Analyzed models will be used most effectively to supplement and corroborate other methods of estimation.

REFERENCES
[7]. www.sepo.spawar.navy.mil/Estimation
[8]. www.rspa.com/reflib/Estimation
[9]. www.sunset.usc.edu/research/COCOTS