CHAPTER 2
CLASSIFICATION OF COST ESTIMATION MODELS

2.1 INTRODUCTION

A number of estimation techniques have been developed and they have the following attributes [8] in common:

- Project scope must be established well in advance.
- Software metrics are used as a basis from which estimates are made.
- The project is broken into modules which are estimated individually.

To achieve reliable cost and schedule estimates, a number of options arise:

- Delay estimation until a later stage in the project.
- Use simple decomposition techniques to generate project costs and schedule estimates.
- Develop empirical models for estimation.
- Acquire one or more automated estimation tools.

The longer we wait, the more we know, and the more we know, the less likely are we to make serious errors in our estimates [187]. In the last chapter we have gone through a brief history in this field. In this chapter we present a classification of various cost estimation models.
2.2 CLASSIFICATION OF COST ESTIMATION MODELS

2.2.1 Classification as per Boehm (1981)

In his book, *Software Engineering Economics* [37], Barry Boehm lists 7 categories of methods used to estimate software costs. They are:

- Algorithmic Models
- Expert Judgment
- Analogy
- Parkinson method
- Price to Win
- Top-Down
- Bottom-Up

*Algorithmic models*

Algorithmic models provide one or more mathematical algorithms, which produce a software cost estimate as a function of a number of variables considered to be major cost drivers.
Expert Judgment

Expert judgment techniques involve consulting with one or more experts, who use their experience and understanding of the proposed project to arrive at an estimate of its costs. Estimates are made by domain experts rather than by estimation experts.

Analogy

Estimation by analogy involves comparing the current project with a database of completed projects in order to find the most similar one.

Parkinson

Parkinson’s Law says, “Work expands to fill the available volume.” This principle is invoked to equate the cost estimate to the available resources. For example, if the project must be completed in 6 months and we have 10 people available, an estimate based on Parkinson’s Law will be 60 person-months.

Price-to-Win

The cost estimate developed by this method is equated to the price believed necessary to win the job, or the schedule believed necessary to be first in the market with a new product, etc.
Top-Down

In top-down estimating, an overall cost estimate for the project is derived from the global properties of the software product. The total cost is then split up among the various components.

Bottom-Up

In the usual bottom-up estimate, each component of the software job is separately estimated, and the results aggregated to produce an estimate for the overall job. Bottom-up estimating can be done in conjunction with any of the methods discussed above.

2.2.2 CLASSIFICATION AS PER BOEHM AND OTHERS (2000)

In 2000, Barry Boehm and others [41] present the following classification of existing estimation techniques:

- Model-based
- Expertise-based (e.g. Delphi, Work Breakdown Structure)
- Learning-oriented
- Dynamics-based
- Regression-based
- Composite-Bayesian
2.2.2.1 MODEL BASED

A model may be static or dynamic. In a static model, a unique variable (say, size) is taken as a key element for calculating all others (say cost, time). In a dynamic model, all variables are interdependent and there is no basic variable as in the static model. When a model uses a single basic variable to calculate all others it is said to be a single variable model. When several variables are used it is said to be a multivariable model. The variable single or multiple that predicts the behavior of software development is called predictors. 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 formulae 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 a 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 [37], Putman's SLIM[191], and Albrecht's' function points [21]. Initially these seem 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 is the time spent calibrating a formal model, the more accurate the cost estimates should be. A distinct disadvantage of formal models is the inconsistency of estimates.
Kemerer [133] conducted a study indicating that estimates varied from as much as 85 - 610% between predicted 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 (LOC) are used, this is obviously not known at the start of the project. LOC is also very much dependant on the programming language and programming environment, therefore this is difficult to determine at an early stage in the problem especially as requirements are likely to be sketchy. Despite this LOC 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 function points (FP) proposed by Albrecht [21,22], 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 LOC. 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:

\[
\text{Effort} = p \times S. \quad \text{(2.1)}
\]

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

Most models allow for non-linear relationships by introducing economies or diseconomies of scale. The general formula being:

\[
\text{Effort} = p \times S^c. \quad \text{(2.2)}
\]

A study published by Walston & Felix [227] which consisted of 60 projects at IBM federal systems division concluded that effort could be modeled as:

\[
\text{Effort} = 5.2 \text{ LOC}^{0.91}. \quad \text{(2.3)}
\]

Models could be proprietary, those which are not fully documented or in the public domain (like ESTIMACS, PRICE S and Knowledge Plan,) [79,196,197] or non-proprietary, those that are fully documented (like COCOMO, COCOMO-II, SDC, SLIM, SEER-SEM etc) [37,49,191,223]
2.2.2.2 EXPERT BASED TECHNIQUES

Expert based techniques are useful in the absence of quantified, empirical data. They capture the knowledge and experience of practitioners seasoned within a domain of interest, providing estimates based upon a synthesis of the known outcomes of all the past projects in which the expert participated. The obvious drawback to this method is that an estimate is only as good as the expert's opinion, and there is no way usually to test that opinion until it is too late to correct the damage if that opinion proves wrong. Years of experience do not necessarily translate into high levels of competency. Moreover, even the most highly competent of individuals will sometimes simply guess wrong. Two techniques have been developed which capture expert judgment but that also take steps to mitigate the possibility that the judgment of any one expert will be off. These are the Delphi technique and the Work Breakdown Structure.

Delphi Technique

The Delphi techniques [106] were developed originally as a way of making predictions about future events. More recently, this technique has been used as a means of guiding a group of informed individuals to a consensus of opinion on some issue. Participants are asked to make some assessment regarding an issue, individually in a preliminary round, without consulting the other participants in
the exercise. The first round results are then collected, tabulated, and then returned to each participant for a second round, during which the participants are again asked to make an assessment regarding the same issue, but this time with knowledge of what the other participants did in the first round. The second round usually results in a narrowing of the range in assessments by the group, pointing to some reasonable middle ground regarding the issue of concern. The original Delphi technique avoided group discussion; the Wideband Delphi technique [37] accommodated group discussion between assessment rounds. This is a useful technique for coming to some conclusion regarding an issue when the only information available is based more on “expert opinion” than hard empirical data.

Chulani and Boehm [56] used the technique to estimate software defect introduction and removal rates during various phases of the software development life-cycle. These factors appear in COQUALMO (CONstructive QUALity MOdel), which predicts the residual defect density in terms in number of defect/unit of size [56]. Chulani and Boehm also used the Delphi approach to specify the prior information required for the Bayesian calibration of COCOMO II [55].
Work Breakdown Structure (WBS)

Long a standard of engineering practice in the development of both hardware and software, the WBS is a way of organizing project elements into a hierarchy that simplifies the tasks of budget estimation and control. It helps to determine just exactly what costs are being estimated. Moreover, if probabilities are assigned to the costs associated with each individual element of the hierarchy, an overall expected value can be determined from the bottom up for total project development cost [24]. Expertise comes into play with this method in the determination of the most useful specification of the components within the structure and of those probabilities associates with each component.

Expert based methods are good for unprecedented projects and for participatory estimation, but encounter the expertise-calibration problems discussed above and scalability problems for extensive sensitivity analyses. WBS-based techniques are good for planning and control.

A software WBS actually consists of two hierarchies, one representing the software product itself, and the other representing the activities needed to build that product [37]. The product hierarchy describes the fundamental structure of the software, showing how the various software components fit into the overall
system. The activity hierarchy indicates the activities that may be associated with a given software component.

Aside from helping with estimation, the other major use of the WBS is cost accounting and reporting. Each element of WBS can be assigned its own budget and cost control number, allowing staff to report the amount of time they have spent working on any given project task or component, information that can then be summarized for management budget control purposes.

Finally, if an organization consistently uses a standard WBS for all its projects, over a piece of time it will accrue a very valuable database reflecting its software cost distributions. This data can be used to develop a software cost estimation model tailored to the organization’s own experience and practices.

### 2.2.2.3 LEARNING ORIENTED TECHNIQUES

Learning-oriented techniques include both, some of the oldest as well as newest techniques applied to estimation activities. The former are represented by case studies, among the most traditional of “manual” techniques; neural networks, which attempt to automate improvements in the estimation process by building models that “learn” from previous experience, represent the latter.
Case Studies

Case studies represent an inductive process, whereby estimators and planners try to learn useful general lessons and estimation heuristics by extrapolation from specific examples. They examine in detail elaborate studies describing the environmental conditions and constraints that are obtained during the development of previous software projects, the technical and managerial decisions that were made, and the final successes or failures that resulted.

Shepperd and Schofield [208] did a study comparing the use of analogy with prediction modes based upon stepwise regression analysis for nine datasets (a total of 275 projects), yielding higher accuracies for estimation by analogy. They developed a five-step process for estimation by analogy:

- Identify the data or features to collect
- Agree data definitions and collections mechanisms
- Populate the case base
- Tune the estimation method
- Estimate the effort for a new project
Neural Networks

According to Gray and McDonnell [89], neural network is the most common software estimation model-building technique used as an alternative to mean least squares regression. These are estimation models that can be “trained” using historical data to produce even better results by automatically adjusting their algorithmic parameter values to reduce the delta between known actuals and model predictions. Gray, et al., goes on to describe the most common form of a neural network used in the context of software estimation, a “back propagation trained feed-forward” network.

The development of such a neural model begun by first developing an appropriate layout of neurons, or connection between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked. The weighted estimating functions between the nodes and the specific training algorithm to be used must also be determined. Once the network has been built, the model must be trained by providing it with a set of historical project data inputs and the corresponding known actual values for project schedule and/or cost. The model then iterates on its training algorithm, automatically adjusting the parameters of its estimation functions until the model estimate and the actual value we are within some pre-specified delta. The specification of a delta value is important. Without it, a model could theoretically
become over trained to the known historical data, adjusting its estimation algorithms until it is very good at predicting result for the training data set, but weakening the applicability of those estimation algorithms to a broader set of more general data.

Wittig [77] has reported accuracies of within 10% for a model of this type when used to estimate software development effort, but caution must be exercised when using these models as they are often subject to the same kinds of statistical problems with the training data as are the standard regression techniques used to calibrate more traditional models. In particular, extremely large data sets are needed to accurately train neural networks with intermediate structures of any complexity. Also, for negotiation and sensitivity analysis, the neural networks provide little intuitive support for understanding the sensitivity relationships between cost driver parameters and model results. They encounter similar difficulties for use in planning and control.

2.2.2.4 DYNAMICS-BASED TECHNIQUES

Dynamics-based techniques explicitly acknowledge that software project effort or cost factors change over the duration of the system development; that is, they are dynamic rather than static over time. This is a significant departure from the other techniques highlighted in this chapter, which tend to rely on static models
and prediction based upon snapshots of development situation at a particular moment in time. However, factors like deadlines, staffing levels design requirements, training needs, budget, etc. all fluctuate over the course of development and cause corresponding fluctuations in the productivity of project personnel. This in turn has consequences for the likelihood of a project coming in on schedule and within budget—usually negative. The most prominent dynamic techniques are based upon the system dynamics approach to modeling originated by Jay Forrester nearly forty years ago [78].

*System Dynamics Approach*

System dynamics is a continuous simulation modeling methodology whereby model results and behavior are displayed as graphs of information that change over time. Models are represented as networks modified with positive and negative feedback loops. Elements within the models are expressed as dynamically changing levels or accumulations (the nodes), rates or flows between the levels (the lines connecting the nodes), and information relative to the system that changes over time and dynamically affects the flow rates between the levels (the feedback loops).

Within the last ten years, this technique has been applied successfully in the context of software engineering estimation models. Abdel-Hamid has built
models that will predict changes in project cost, staffing needs and schedule over time, as long as the initial proper values of project development are available to the estimator [95,97,97,99]. He has also applied the technique in the context of software reuse, demonstrating an interesting result. He found that there is an initial beneficial relationship between the reuse of software components and project personnel productivity, since less effort is being spent developing new code. However, over time this benefit diminishes as older reuse components are retired and no replacement components have been written, thus forcing the abandonment of the reuse strategy until enough reusable components have been created, or unless they can be acquired from an outside source. More recently, Madachy used system dynamics to model an inspection-based software lifecycle process [153]. He was able to show that performing software inspections during development slightly increases programming effort, but later decreases effort and schedule during testing and integration. For typical industrial value of this parameter, the savings due to inspections considerably outweigh the costs. Dynamics-based techniques are particularly good for planning and control, but particularly difficult to calibrate.
2.2.2.5 REGRESSION-BASED TECHNIQUES

Regression-based techniques are the most popular ways of building models. These techniques are used in conjunction with model-based techniques and include "Standard" regression, "Robust" regression etc.

"Standard" Regression – Ordinary Least Squares (OLS) method

"Standard" regression refers to the classical statistical approach of general linear regression modeling using least squares. It is based on the Ordinary Least Squarer (OLS) method discussed in many books [130]. The reasons for its popularity include ease of use and simplicity. It is available as an n option in several commercial statistical packages such as Minitab, SPlus, SPSS, etc.

A model using the OLS method can be written as

\[ y_t = \beta_1 + \beta_2 x_{1t} + \ldots + \beta_k x_{kt} + e_t \quad \ldots \quad (2.6) \]

Where

- \( x_{1t}, \ldots, x_{kt} \) are predictor (or regressor) variables for the \( t \)th observation
- \( \beta_2, \ldots, \beta_k \) are response coefficients
- \( \beta_1 \) is an intercept parameter
- \( y_t \) is the response variable for the \( t \)th observation.
The error term, $e_i$, is a random variable with a probability distribution (typically normal).

The OLS method operates by estimating the response coefficients and the intercept parameter by minimizing the least squares error them where $r_i$ is the difference between the observed response and the model predicted response for the $i_{th}$ observation. Thus all observations have an equivalent influence on the model equation. Hence, if there is an outlier in the observations then it will have an undesirable impact on the model.

"Robust" Regression

Robust Regression is an improvement over the standard OLS approach. It alleviates the common problem of outliers observed in software engineering data. Software project data usually have a lot of outliers due to disagreement on the definitions of software metrics, coexistence of several software development processes and the availability of qualitative versus quantitative data.

There are several statistical techniques that fall in the category of 'Robust' Regression. One of the techniques is based on Least Median Squares method and
is very similar to the OLS method described above. The only difference is that this technique reduces the median of all the $r_i^2$.

Another approach than can be classified as “Robust” regression is a technique that uses the data points lying within two (or three) standard deviations of the mean response variable. This method automatically gets rid of outliers and can be used only when there are a sufficient number of observations, so as not to have a significant impact on the degrees of freedom of the model. Although this technique has the flaw of eliminating outliers without direct reasoning, it is still very useful for developing software estimation models with few regressor variable due to lack of complete project data.

Most existing parametric cost models (COCOMO II, SLIM, Checkpoint etc.)[191,223] use some from of regression-based technique due to their simplicity and wide acceptance.

2.2.2.6 Composite Techniques

As discussed above there are many pros and cons of using each of the existing techniques for cost estimation. Composite techniques incorporate a combination
of two or more techniques to formulate the most appropriate functional form for estimation.

**Bayesian Approach**

An attractive estimating approach that has been used for the development of the COCOMO II model is Bayesian analysis [55].

Bayesian analysis is a mode of inductive reasoning that has been used in many scientific disciplines. A distinctive feature of the Bayesian approach is that it permits the investigator to use both sample (data) and prior (expert-judgment) information in a logically consistent manner in making inferences. This is done by using Bayes’ theorem to produce a ‘post-data’ or posterior distribution for the model parameters. Using Bayes’ theorem, prior (on initial) values are transformation can be viewed as a learning process. The posterior distribution is determined by the variances of the prior and sample information. If the variance of the prior information is smaller than the variance of the sampling information, then a higher weight is assigned to the prior information. On the other hand, if the variance of sample information is smaller than the variance of the prior information, then a higher weight is assigned to the sample information causing the posterior estimate to be closer to the sample information.
Bayesian analysis has all the advantages of "Standard" regression and it includes prior knowledge of experts. It attempts to reduce the risks associated with imperfect data gathering. Software engineering data are usually scarce and incomplete and estimators are faced with the challenge of making good decisions using this data. Classical statistical techniques described earlier derive conclusions based on the available data. But to make the best decision it is imperative that in addition to the available sample data we should incorporate nonsample or prior information that is relevant. Usually a lot of good expert judgment based information on software processes and the impact of several parameters on effort, cost, schedule, quality etc. is available. This information doesn’t necessarily get derived from statistical investigation and hence classical statistical techniques such as OLS do not incorporate it into the decision making process. Bayesian techniques make best use of relevant prior information along with collected sample data in the decision making process to develop a stronger model.
2.2.3 CLASSIFICATION ACCORDING TO WIECZOREK (2001)

Wieczorek provides definitions of terms like *estimation model*, *estimation method* and *estimation technique*. Her classification of cost estimation methods is based on these definitions, and is shown in Figure 2.1.

There are two definitions of an *estimation method*:

I. An *estimation method* consists of one or more *estimation techniques*. An *estimation technique* is a procedure to devise a cost expenditure estimate directly from available expert knowledge and/or project data. An *estimate* is obtained by applying one or more *estimation techniques* directly on available data or knowledge.

II. An *estimation method* consists of one or several *estimation models*, possibly a *modeling method* and an *application method* to apply the *estimation model(s)*. An *estimation model* is built from one or more *modeling techniques* according to a *modeling method*. A *modeling technique* is a single procedure to construct an *estimation model* from available data and/or expert knowledge. An *estimate* is then obtained by applying one or more *estimation models* in a specific context, according to a *model application method*.
Figure 2.1 – Classification of cost estimation methods (Wieczorek 2001)

2.3 COMPARISON OF SOFTWARE COST ESTIMATION METHODS

Different models as classified by Boehm (1981) initially are compared in this section. The table below shows a comparison of the strengths and weaknesses of the software cost estimation methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
</table>
| Algorithmic Model | • Objective, repeatable, analyzable formula  
<p>|                  | • Efficient, good for sensitivity analysis    | • Subjective inputs                             |
|                  | • Objectively calibrated to experience         | • Unable to deal with exceptional circumstances |
|                  |                                               | • Need to calibrate to local circumstances      |
|                  |                                               | • Calibrated to past, not future                |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| **Expert Judgment**         | • Able to factor in differences between past project experiences and new project  
                              | • Factor in interactions, and exceptional circumstances             | • Judgment may be no better than the estimator  
                              |                                                                           | • May have incomplete recall of previous projects  
                              |                                                                           | • May have biases                                                          |
| **Estimation by Analogy**   | • Based on actual experience of a project                            | • Difficult to find out the degree that the previous project is representative of the new project |
| **Parkinsonian Estimation** | • Generally correlates with some experience                          | • Usually not accurate  
                              |                                                                           | • Generally produces large overruns  
                              |                                                                           | • Reinforces poor practice                                                  |
| **Price to Win**            | • Often gets the contract                                           | • Based on the customer's budget or marketing, not on the functionality of the product  
                              |                                                                           | • Reinforces poor software development practice                             |
| **Top-down Estimation**     | • System level focus  
                              | • Efficient                                                           | • Less detailed basis, may miss some components  
                              |                                                                           | • Less stable than a multi-component estimate                                |
| **Bottom-up Estimation**    | • More detailed basis  
                              | • Fosters individual commitment, since each estimate is backed by the individual responsible for the job | • May overlook system level costs i.e. integration, configuration management  
                              |                                                                           | • Requires more effort to find out the components                           |
| **Machine Learning**        | • Takes into account past experience  
                              | • Uses a reasoning approach  
                              | • More robust against noise and data outliers                           | • Process may not be visible to the user  
                              |                                                                           | • Some methods are not suitable for small datasets                           |
Table 2.1: Strengths and Weaknesses of the Software Cost Estimation Methods[37]

It is important to realize that none of the alternative methods is better than the others from all aspects. The Parkinson and Price-to-Win methods are unacceptable and do not produce sound cost estimates, however, the strengths and weaknesses of the other techniques are complementary. For example, algorithmic is complementary to expert judgment, and top-down is complementary to bottom-up. It is important to use a combination of techniques, and to compare and iterate the estimates obtained from each. This ensures a reliable cost estimate. The combination chosen will depend on the cost-estimation objectives defined. Boehm [37] recommended that an effective combination often used is:

- Top-down estimate using the judgment of more than one expert, using analogy estimation where a comparable previous project is available
- Bottom-up estimate using an algorithmic model, with inputs and component-level estimates provided by future performers and a comparison and iteration of both estimates.
2.4 CONCLUSION

Cost estimation models are not a substitute for a detailed estimate by task by project management. It is important to remember that software cost estimations are an aid to project planning, and that good project planning helps to meet software cost estimations. Cost estimation models highly depend on the user's knowledge of application domain, analysis ability, and the understanding of the cost model itself. Cost estimation models often take a considerable amount of time, effort, and knowledge to obtain useful results. The process of meeting the estimate is equally critical to the process of making the estimate. It is important to remember that all of the models make the assumption that the inputs to the cost estimation model will be accurate, and that the project will enjoy good management by both the developer and the customer.