CHAPTER 1

BRIEF HISTORY AND LITERATURE SURVEY

1.1 INTRODUCTION

Software engineering as a discipline emerged in response to the 'software crisis' perceived by the industry. Therefore, the need to run the project smoothly cannot be understated. Effective software management is becoming increasingly important to produce good quality software, that is maintainable and within time schedule. New methods and techniques are required to help the project manager to plan, monitor and control the software processes and products. Software measurement, when integrated into the overall management process will provide the necessary information to the project manager to identify the risks, track specific problems and estimate effort, cost & schedule. This may also develop alternate solutions and select the best approach for solving the problems. Software engineering as defined by IEEE [118] is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software that is the application of engineering principles to software development. Software measurement provides an insight to the manager who needs to make the decisions critical to the success of software.
1.2 SOFTWARE METRICS

‘You cannot control what you cannot measure’[70]. This fundamental reality underlies the importance of software metrics. Software metrics can be defined as [85] “The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information with the use of those techniques to improve that process and its product. Software metrics are all about numbers and the same are applicable to the whole development life cycle right from initiation, to monitoring reliability of the end product. Software metrics improve the process of developing software and improve all aspects of the management of that process. [75, 76, 86]

Some metrics like size, effort, cost, maintainability and understandability are considered in this thesis and they are evaluated using a soft computing approach. The most important software metrics that need to be evaluated are size, effort and cost in a program, besides maintainability and understandability.

1.3 SOFTWARE COST ESTIMATION

As software becomes a larger part of projects today, it is becoming more and more critical to accurately determine the effort and cost involved to produce the software product. With this understanding, we can plan appropriately to ensure the project is completed on time and within budget. [8, 11, 211]
Other disciplines like mechanical and electrical engineering can more accurately estimate the costs associated with a project. The differences between cost estimation in other disciplines and in software engineering are:

- Source instructions are not a uniform commodity.
- Software requires the creativity and cooperation of human beings, whose behavior is difficult to predict.
- Software has a much smaller base of relevant quantitative historical experience.
- Every software project is a new project with different problems.
1.3.1 DEFINITIONS

Estimate a concrete value or distribution that reflects the most knowledgeable statement that can be made at a point in time about the costs needed for future or ongoing project.

Effort is measured in person-days, person-months or person-years and is the usual outcome of a cost estimation process. It determines the amount of work required to complete the estimated project.

Duration, or elapsed time, or schedule, is usually measured in months or years, and determines the amount of calendar time required to finish the estimated project. This is not a simple function of effort divided by manpower because the application of manpower does not affect the duration linearly [49].

Cost drivers, or cost factors, are dependent variables that have been identified through statistical analysis of project attributes and are considered to influence the effort. Cost drivers are used as input to a number of cost estimation models. The most common cost driver is software size.
**Software Size** is a measure of the size of the software to be made. The two most popular size metrics are physical size measured in lines-of-code and functional size measured in function points.

### 1.3.2 TERMINOLOGY

**Model or estimation model:** An unambiguous, reusable representation of the relationship between cost expenditures (development effort, cost, productivity) and its most important driving factors (cost drivers).

**Modeling technique:** A procedure to construct a model from available data and/or expert knowledge.

**Modeling method:** The application of one or more modeling techniques according to some guidelines on how to combine them to build a cost estimation model.

**Model application method:** A procedure specifying how to obtain an estimate by applying one or several models in a specific context.
**Estimation technique:** A procedure to devise a cost expenditure estimate directly from available expert knowledge and/or project data. No model building is involved.

**Estimation method:** It consists of either (A) one or more estimation techniques, or (B) one or several models, possibly a modeling method and a method to apply the model(s) (application method). This distinction is made because some estimation methods do not include any explicit cost modeling method. The estimates are solely based on expert judgment.

### 1.3.3 THE MOST COMMON MISTAKES IN SOFTWARE COST ESTIMATION

There are several mistakes, which are commonly made in software cost estimation. [186, 187] These include:

- Underestimating the amount of time and effort required.
- Imprecise and drifting requirements. Requirements creep increases the time and effort required for the project, so the cost estimation needs to be changed accordingly.
• Not recognizing that projects are different from the previous projects. Often previous project's estimates are used verbatim without properly considering the differences.

• Estimates forced to match resources available.

• Human bias. Certain estimators may have a bias as to how long and how much projects will cost?

1.3.4 DIFFICULTIES IN COST ESTIMATION

There are many difficulties in cost estimation, and some can be listed as follows [94]:

• There is a lack of data on completed software projects, which could support project managers in making estimates.

• Estimates are made in a hurry. Often estimates are needed before system requirements are produced, causing estimators to write an estimate too quickly for a system they do not fully understand.

• Clear, complete and reliable specifications are difficult to formulate.

• Characteristics of software and software development make estimating difficult.

• A great number of factors (cost drivers) have an influence on the effort and time to develop software. These cost drivers are difficult to determine in operation.
• Rapid changes in information technology and the methodology of software development.

• There is an apparent bias of software developers towards underestimation.

• The estimator may ignore the fact that a lot of work will be done by less experienced people with lower productivity rate than himself.

• There exists a serious misassumption of a linear relation between the required capacity per unit of time and the available time. According to Brooks [49], this is seriously wrong.

• The estimator tends to reduce the estimate to some degree, in order to make the bid more acceptable.

• The software industry is usually developing new products. In other industries it is more usual to produce the same product over and over.

1.3.5 STEPS IN SOFTWARE COST ESTIMATION

Software project costs are the sum of all the costs in the software project. This may include hardware costs, travel costs, training costs, effort costs, etc. Effort costs are generally the largest. Therefore, most cost estimations estimate the effort cost using person-months (PM) as cost unit.

Boehm has outlined seven general steps to doing good software cost estimation [37]. These are:
1. Establish objectives
2. Plan for required data and resources
3. List software requirements
4. Work out as much detail as feasible
5. Use several independent techniques and sources
6. Compare and iterate estimates
7. Follow up

A lot of work has been done in this field and the next section presents a brief history.

1.4 BRIEF HISTORY

In the early 1960s, the frequency and magnitude of software cost overruns was becoming a critical problem. There was a need for more accurate software cost estimation techniques. This lead to the study of software cost estimation. For several projects at IBM, in the late fifties Norden plotted manpower frequency distributions, in which he showed how many people were allocated to the development and maintenance of a software product during it’s life-cycle. The curves he came up with, happened to be very similar to the Rayleigh curves that were used to describe the distribution of other phenomena than manpower as a
function of time. Later, the Norden-Rayleigh curves became the basis for several software cost estimation models (Putnam and Jensen) [123, 191].

Through the late 1950s and early 1960s, little progress was made in the area of software cost estimation research. In 1964, the U.S Air Force and System Development Corporation (SDC) started working on a landmark project in the field of software cost estimation. SDC collected 104 attributes (cost drivers) of 169 software projects. The equations were derived by statistical analysis from the total set of 169 projects, as well as of selected subsets from the total set. The most conclusive result from the SDC study was that there were too many non-linear aspects of software development for a linear cost estimation model to work very well. However, the SDC model provided a valuable base of information and insight for cost estimation, and it also became a foundation for several other cost estimation models. Also worth mentioning is the PRICE H [79] model developed by Frank Freiman, which was later, extended for software product. (PRICES).

The 1970s turned out to be a very active decade. In the 1970s, researchers identified the major cost factors contributing to software development costs. The most well known model of this type is the Constructive Cost Model (COCOMO), developed in the late 1970s by Barry W. Boehm and described in his classic book _Software Engineering Economics_ [37].
Besides COCOMO, some of the more notable were TRW Wolverine [229], the Doty model [101], IBM-FSD[227], the Putnam model[191], the RCA PRICE S model[79], the Boeing model, and a series of models developed by GRC. At the end of the decade A. Albrecht and J. Gaffney of IBM developed Function Point Analysis (FPA) to estimate the size and development effort for management information systems[21]. This marked the beginning of a completely new way of measuring software size. The first two commercial software cost estimation tools emerged at the end of the 1970s. Frank Freiman of RCA developed PRICE S in 1977 and Larry Putnam of QSM developed SLIM in 1979.

During the 1980s, Personal Computers (PCs) came into general use and several tools that implemented COCOMO appeared. Robert C. Tausworthe extended the work of Boehm (COCOMO)[37], Herd (Doty)[101], Putnam (Putnam SLIM)[191], Walston-Felix (IBM-FSD)[227] and Wolverton (TRW Wolverton)[229] to develop a cost model for NASA's Jet Propulsion Laboratory, known as SOFTCOST or Deep Space Network (Tausworthe 1981)[219].

Randall W. Jensen extended the work of Putnam by eliminating some of the undesirable behavior of Putnam's SLIM. The Jensen model, which was also influenced by COCOMO and the Doty model, became the basis of a series of commercial estimating tools in the 1980s, including the Jensen System 1, JS2,
JS3, and the SEER-SEM products. John W. Bailey and Victor R. Basili [30] based their work on the work of Boehm and Walston-Felix, and presented a model generation process for the development of a resource estimation model for any particular organization. They assumed that the coefficients in any effort equation would be highly dependent on the environment and personnel at a particular installation, Hence coefficients derived from a local database would lead to a much more accurate model.

Some of the other notable models of the 1980s were COPMO (Conte et al. 1986)[61], BYL, SPQR/20 (Jones 1986)[127] and ESTIMACS (Rubin 1983). In 1986, Capers Jones developed an alternative form of function point called feature points (Jones 1996)[126]. Feature points were designed to include the effect of computationally complex algorithms on development costs, and were primarily intended for use with real-time, embedded and systems software. Another revision of FPA was proposed by Charles Symons, who developed the Mark II Function Points [215] to overcome what he considered to be weaknesses in Albrecht’s approach to Function Point Analysis. In 1982, Tom DeMarco published his classic book Controlling Software Projects,[70] which contained an independent form of functional metric that, by coincidence, duplicated some of the features of Albrecht’s function points. The DeMarco function point[70], also known as bang metric, measures the amount of functionality by counting certain components of a structured analysis specification model. Several features of the
original DeMarco function point have turned up in a recent metric called Boeing 3D function points, as well as several other function point variants. The usage and popularity of function points grew so rapidly that the non-profit International Function Point Users Group (IFPUG)[207] was founded in order to develop standards for function point counting.

Until the mid 1990s, the main focus of software cost estimation research was on algorithmic models. The lack of consistent results caused researchers to look at alternative techniques, such as estimation by analogy and various machine learning techniques. Estimation by analogy received a lot of attention and a number of promising tools such as ESTOR [168,169] and ANGEL [206] were developed. Another notable event during this period was the release of COCOMO 2.0 [49,223] that was developed to address limitations in the original COCOMO.

After going through the history in the field of software size, effort and cost estimation a literature survey was carried out, which is presented in this chapter.

1.5 LITERATURE SURVEY

A literature survey was carried out. Notable papers/articles in the field are as follows:
Boehm [37] summarizes the state-of-the-art and recent trends in software cost engineering economics. His paper surveys the field of software cost estimation, including the major estimation techniques available, the state-of-the-art in algorithmic cost models, and lists the outstanding research issues in software cost estimation. The algorithmic cost models presented include Wolverton[229], Putnam SLIM[191], Doty[101], PRICE S[79] and COCOMO[37].

Kemmerer [133] presents an empirical validation of four algorithmic models, SLIM, COCOMO, Function Point Analysis and ESTIMACS. Kemmerer seeks answers to three main questions: (1) Are cost estimation models generalizable to environments other than that in which they were developed, and if not, can they be easily calibrated? (2) Are models that do not use source lines of code as an input as accurate as those that do? (3) Are the models available in the open literature as accurate as proprietary models?

Haemstra [91] gives an overview of the state-of-the-art of software cost estimation. The models covered are COCOMO, FPA, PRICE S, Putnam, BYL, ESTIMACS, SPQR-20 and BIS-Estimator. He also provides a chronological and extensive list of models and tools.

Fairley [73] presents a framework for characterizing the ways in which various estimation techniques model the relationships and trade off possibilities among
schedule, budget, assets, product objectives and constraints. Boehm and others [41] summarizes several classes of software cost estimation models and techniques. They focus on the classification of existing techniques into six major categories; model-based, expertise-based, learning-oriented, dynamics-based, regression-based and composite. They also conclude that experience to date indicates that neural network - and dynamics-based techniques are less mature than the other classes of techniques. According to the authors, no single technique is best for all situations. In order to get the best estimates, they recommend a careful comparison of the results of several approaches.

A. Albrecht & J. E. Jeffeny [21] proposed the function point approach to prediction of effort.

L. Putnam’s [191] paper provides an empirical solution to the macro software sizing and estimation problem. S. Chulani, B. Boem & B. Steece’s [57] paper calibrates software cost models using Bayesian Analysis. L. Briand & I. Wieczorek’s [47] paper investigates two essential questions related to data driven software cost modeling-1) what modeling techniques are likely to yield more accurate results and 2) what are the benefits and drawbacks of using organization specific data as compared to multiorganization databases?

O. Doban, A. Pataricza [68] detail the experiments of a pseudo cost estimation with a special emphasis on cost estimation driven system design. C. Symon’s
[215] paper describes the principles of Mark II approach, calibration of Mark II approach and on validity and application of FPA.

Since it was proposed to use the soft computing approach to software metrics, several papers applying the approach to software engineering applications were also surveyed. The notable amongst them are listed below:

G. Finnie & G, Wittig [77] examine the performance of back propagation artificial neural networks in estimating software development effort and the potential of using case based reasoning. W. Pedrycz [180] highlights a number of selected and most visible trends occurring at the junction of Computational Intelligence and Software Engineering. J. Ryder [199] investigates the application of Fuzzy modeling techniques to two of the most widely used software effort prediction models, the COCOMO and the FPA method. W. Pedrycz, J. Peters, S. Rammana [181 ] use the fuzzy sets in the software cost estimation problem.C. Yau & R. Tsoi [231] present the experiences of using fuzzified function Point Approach to estimate the software size through a case study and compare it with the conventional FPA.

A.Idri, A. Abran & T. Khoshgaftaar [115] use the new approach based on reasoning by analogy, fuzzy logic and linguistic quantifiers to estimate software project effort. G. Boetticher [43] has dealt with how to deal with scarcity of empirical data in software engineering discipline and how to generate sufficient amount of data. A. Idri, A. Abran & T. Khoshgaftaar [114 ], G. Boetticher, K.
Srinivas, D. Eichmann [44] & G. Boetticher[42] have explained the possibility of using an alternative neural network approach to effort & cost estimation. X. Huang, D. Ho, J. Ren & L. F. Capretz [113] and A. Hodgkin son, Paul W. Grrrat[108] have proposed a neuro fuzzy cost estimator. G. Cbenko [65], K. Funahashi [80], A. R. Barron [29], Hornik and others [110] have shown that a multiplayer feed forward neural network using back propagation algorithm can be used as a universal approximator. K. K. Aggarwal, Y. Singh & J. Chhabra [10,16] have shown how the fuzzy approach could be used to provide a unified measure of software maintainability and understandability.

1.6 ORGANIZATION OF THE THESIS

This chapter of the thesis presented an introduction to software estimation and the difficulties in estimation. It presented an overview of the work done by various researches in the field of software cost estimation since the early 60’s till date. It presented the literature survey that was carried out where the thrust has been on size/effort/cost estimation techniques and models as well as measurement of maintainability and understandability. Further, literature survey was also carried out in the field of the soft computing approach (neural network models, fuzzy models and probabilistic reasoning) being used in software engineering application particularly in estimating size, effort, cost, maintainability and understandability.
Chapter 2 of the thesis reviews various cost estimation models and classifies them. It further, compares the various models, discussing the strengths and weaknesses of each model.

Chapter 3 of the thesis introduces the soft computing approach. It discusses that the neural network model and fuzzy model could be used for measurement of software metrics. It further proposes hybrid models which combine two or more paradigms of computing to arrive at better solutions.

Chapter 4 discusses the Function Point Analysis method, which is independent of the language tools, or methodologies used for implementation and propose a fuzzy model to capture the inherent fuzziness that is present in all model measurement parameters i.e. all inputs and complexity factors.

Chapter 5 deals with estimating of LOC where a neural network is proposed for estimating the Line of Code once the function point count, the FP standard used, the language used and team size are known. This model is validated by using data of 88 projects available from ISBSG (release 9).

Chapter 6 explores the possibility of estimating Lines of Code, when only Function Point count is known. Models are proposed in this chapter that use simple linear regression techniques, neural networks and a combination of the two. This chapter also proposes an expert committee model, which is a combination of robust regression, and neural network model. All the models
proposed in the chapter including the expert committee model are validated by project data (release 9) from ISBSG.

Chapter 7 presents estimation of effort using a neural network model.

The process of changing software after it has been delivered and is in use is called software maintenance. Maintenance is reported to be the costliest phase. Cost of maintenance can be up to 65-75% of total cost of software. A good measure of maintainability can help manage the software maintenance phase better. It cannot be measured adequately by only source code or documents.

Chapter 8 of the thesis provides a new unified measure of software maintainability. The four parameters of which maintainability is an integrated measure are average live variables in a program, the average life span of variables, the comment ratio, which is a measure of documentation quality and average cyclomatic complexity. The model proposed here is a fuzzy model which is validated by project data for all parameters, as well as corrective maintenance time. Empirical results prove that integrated measure of maintenance proposed by this fuzzy model showed strong co-relation to the average the corrective maintenance time.

Software is a collection of various documents and source code. Understandability of the software is one aspect, which has a lot of impact on maintainability, and requires a good comprehension of logic of the source code. An integrated measure of software understandability, which depends on total spatial complexity,
documentation quality and cohesiveness source code and documents, is provided in Chapter 9 of this thesis. A neural network model which can handle the subjectivity of their individual contribution towards understandability is used.

Fuzzy as neural models have been proposed for measuring various software metrics in this thesis. To find out which model is better, a sensitivity analysis of the two models was carried out. This is presented in Chapter 10.

Chapter 11 presents the conclusion and the scope of future work.