1996

Towards a model for software project estimating

Stuart Hope
Edith Cowan University

Towards A Model for Software Project Estimating

Stuart Hope

1996

Towards A Model for Software Project Estimating

By Stuart Hope B.App.Sc.

A Thesis Submitted in Partial Fulfilment of the Requirements for the Award of
Bachelor of Science (Honours) Computer Science
at the Computer Science Department
Faculty of Science, Technology and Engineering
Edith Cowan University

Date of Submission: 19th July 1996

Table of Contents

Abstract

The use and development of software is an integral and critical part of modern industrial society. The outcomes of many software development and maintenance projects have been less than satisfactory with significant numbers being over schedule, lacking in functionality and over budget. These problems are the result of poor management of both the process and the product.

One of the major problems to overcome in the management of software development projects is the ability to predict the outcomes early in the project when there are a large number of unknowns. The ability to reliably predict the outcomes in a repeatable manner requires accurate estimating techniques that are theoretically sound, practical to use, relevant to the current situation and can cope with all the project variables. Whilst a number of estimating techniques have been developed they are poor in their predictive abilities, do not to take a total project approach and are not used by practitioners.

This proposal is to define a model that will build on the strengths of the current estimating techniques, account for their weaknesses and provide a framework for the development of practical techniques that encompass all aspects of a software development project.

Declaration

"I certify that this thesis does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any institution of higher education and that, to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where due reference is made in the text".

Stuart Hope
19th July 1996

Acknowledgments

I would like to thank my supervisor, Ken Mullin, for his comments on the draft documents. I would also especially like to thank Julian Terry for his encouragement and support in the preparation of this work.

Figures

Tables

1. Introduction

1.1 The Background to the Study

Software systems are now ubiquitous. Software impacts on virtually all aspects of modern industrial society and is economically critical. Software is used to teach, educate, govern, manage, entertain and manufacture. Most electrical and mechanical equipment now includes software, in part, to provide control and functionality. The effective functioning of modern society is becoming increasingly dependant on the production of cost effective software.

Software projects tend to be at the top end of complexity in human endeavours. In most industries it is normal to produce the same type of products repetitively. However the development of software tends to be the continuous design and production of new artefacts using new tools and methods. It is interesting to note that with most human activities that are new or novel in nature it is difficult to predict the outcomes. This has been so in all industries and is of particular significance in software development as each project is a new design exercise. As a consequence of this failure to deliver the expected outcomes numerous authors have referred to it as the "software crisis". Pressman (1992) prefers to call it a "chronic affliction" because the problems in the industry have been causing pain and distress for a long time and it appears they will continue indefinitely.

The construction of software systems is dynamic with a large number of variables affecting its outcome. Some of the variables are known and others are not when the most critical estimates are required to be made at project initiation. As a consequence software projects experience a high rate of failure because their success criteria is judged on highly suspect initial estimates. They constantly fail to meet their financial, schedule, effort, functional and quality targets. There is a school of thought, Thomsett (1991), that with any reasonable sized development a project can only meet one or two of the above targets. Software engineering is a new field of human endeavour whose knowledge base is low on how to effectively measure the attributes and entities that contribute to the building of systems. The demands and the environment, both in terms of the requirements expressed and the enabling technology are changing and evolving rapidly.

What are required are some methods to improve our ability to work in such an environment and increase the probability of being successful in the delivery of software systems. Estimating is one of the key Software Engineering techniques that will enable the rationalisation of decision-making regarding software development. More accurate estimates will increase the probability of success. Techniques are also required that provide a step-wise feedback mechanism to enhance the accuracy of estimates as the projects proceed (Abdel-Hamid, 1993).

A full practical estimating model is an ambitious goal that will require significant empirical studies and experiments together with input from practitioners and researchers in order to provide validation. The intention of this research is to provide a comprehensive model that takes a total software project approach and act as a foundation to be modified, extended and perhaps refuted. Most current estimating techniques only consider a sub-set of the total costs and effort involved.

1.2 The Significance of the Study

Software is critical for the future of Australian industry.

Pressman (1992) asserts that planning is one of the pivotal activities in the software development process and good estimates are a precursor to good planning.

Most of the crises in the industry can be attributed to an inability to manage (Weinberg 1993). A key input into the management and planning process is an estimate of the cost, schedule and effort of the work to be performed.

1.3 The Purpose of the Study

This research aims to develop a model that comprehensively deals with all the recognised complexities of estimating software development and maintenance and hence to provide an effective way of managing projects. Its purpose is to investigate current software project estimating techniques, establish their degree of validity and develop a model that overcomes their perceived weaknesses.

1.4 Research Questions

The questions that this research will try to answer are :-

- What are the strengths and weaknesses of current software estimating techniques?
- What are the common features of existing software estimating techniques?

- What are the barriers to the industrial use of estimating techniques? Surveys have shown that techniques are not used widely. Hihn & Habib-agahi (1991) showed that only 7% of respondents to their survey used models. It is of little use in devising techniques unless they are of practical benefit and hence an understanding of the barriers to use must be understood. Park, Goethert & Webb (1994) conducted a survey that looked at the needs and improvements required in software cost estimating.
- Can an optimal model be created that includes the strengths of existing models and also overcomes their weaknesses? By optimal the model must be comprehensive, theoretically sound and relatively easy to use in practice - i.e. techniques can be derived from the model that can be used easily by practitioners.

2. Method

The work proceeded by:

1. A detailed examination of existing techniques to determine :
    - theoretical strengths and weaknesses;
    - commonality of entities and attributes;
    - explicit and implicit assumptions;
    - inclusivity of the techniques;
    - practical strengths and weaknesses.
2. Analysis of two existing projects to determine:
    - a classification of the project types;
    - methods and techniques used in estimate formulation;
    - accuracy of the above techniques;
    - identification of "gaps" in the techniques where inaccurate through exclusion where major cost elements in a project were not catered for by the estimating technique.

    The subject in the project examination was a semi-government utility who had a considerable base of project information. Whilst it is recognised that the information obtained is subjective in areas and not statistically valid, the projects however form a representative sample that highlight some of the estimating difficulties that are encountered in practice. Also the result of this research is not intended to be definitive but a pointer to future work.

3. Analysis of published surveys of industrial organisations to determine:
    - utilisation of existing techniques;
    - perceived strengths and weaknesses of existing techniques;
    - barriers to the use of existing techniques;
    - desired attributes of an estimating technique.

    Information relating to estimating technique utilisation was obtained from two published surveys, one conducted in the USA and the other in New Zealand (Hihn & Habib-agahi, 1991: Wydenbach & Paynter, 1995).
4. Synthesis of the data into a model, designed to overcome weaknesses of existing techniques and their utilisation, capitalise on strengths and cater for perceived "gaps".

3. Review of the Literature

3.1 General

The history and general classification of the estimating techniques or methods will be discussed and then a detailed examination of the more prevalent techniques will be given.

The most widely quoted work in estimating is Boehm (1981) who was the first to categorise estimating techniques into algorithmic models, expert judgement, analogy, decomposition, Parkinson and "Price to Win". The later two techniques are not really estimating techniques but a recognition of reality and expediency in some organisations. More recently Humphrey (1995) has extended this list to include his own technique and Putnam's Fuzzy Logic. Putnam & Myers (1992) do not elaborate the Fuzzy Logic technique, however they do provide some useful information that can be incorporated into an estimating database.

From the literature surveyed the most widely reported and used formal techniques are COCOMO and Function Point Analysis. These are considered formal because they have a well documented model with repeatable processes and methods by which estimates are calculated. These techniques are discussed in mor detail below. The other techniques such as estimating by analogy are not formally described in the software industry and hence would vary widely from practitioner to practitioner.

The formulation of any software metric must be defined with its intended use in mind. That is, without the clear specification of goals the metric is to achieve the measures will be of little practical benefit. This view is espoused by Fenton (1991) and Gilb (1988) who support Basili's Goal Question Metric approach to measurement (Basili & Rombach, 1988). Daskalantonakis (1992) provides practical experiences with this approach.

Whilst some work, such as Mukhopadhyay & Kekre (1992), has been published that addresses some of the issues involved with software estimating, few with the exception of Kitchenham, Pfleeger & Fenton (1995) have addressed the fundamental theoretical issues that form a necessary scientific basis for any technique. Matson, Barrett & Mellichamp (1994) provides an assessment through the use of several statistical models that relate software development effort to software size in terms of function points. They are concerned with the empirical data upon which the models are based and the lack of attention to the aptness of the models. Jorgensen (1995) in addressing issues relating to the prediction of maintenance effort concludes, after the examination of several prediction models, that "a formal prediction model should not replace the use of expert predictions". This would support Boehm's (1981) Wideband Delphi approach.

3.2 Function Point Analysis - Albrecht

Function Points were devised by Albrecht and first published in 1979 (Albrecht, 1979). Jones (1991) reports the goals set for this measure were that:-

- it dealt with the external features of the software that were important to the user,

- it could be applied early in a product's lifecycle,
- it could be linked to productivity and
- be independent of the coding language.

Various modifications have been made to Function Points including Symonds Mark II Function Point metric and Jones' Feature Points. Both of these techniques are discussed below. These modifications came about because of perceived weaknesses such as not accounting for algorithmic complexity. Dreger (1989) was instrumental in making this estimating measure available to the general public with his publication, which was essentially a function point tutorial. Garmus & Herron (1996) is probably the most recent publication that provides function point counting guidance which includes examples for the counting of Graphical User Interface applications.

Function Points measure software by quantifying the functionality provided to the user based primarily on logical design. The objectives of function point counting are to :-

- Measure functionality that the user requests and receives
- Measure software development and maintenance independently of the technology used for implementation.

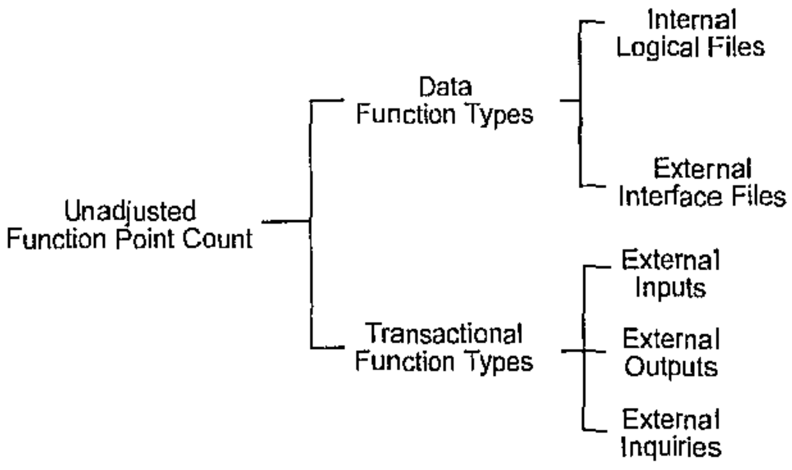
There are three types of function point counts. These being :-

- Development project function point count
- Enhancement project function point count

- Application function point count

The unadjusted function point count reflects the specific countable functionality provided to the user by the project or application. The application's specific user functionality is evaluated in terms of what is delivered by the application, not how it is delivered. Only user-requested and defined components are counted. The unadjusted function point count has two function types - data and transactional. The composition of these function types are shown in Figure 1.

Figure 1 Composition of Function Points



Data function types represent the functionality provided to the user to meet internal and external data requirements. Data function types are either internal logical files or external interface files.

- An internal logical file (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application being counted.

- An external interface file (EIF) is a user identifiable group of logically related data or control information maintained outside the boundary of the application being counted.

Transactional function types represent the functionality provided to the user to process data by an application. Transactional function types are defined as external inputs, external outputs and external inquiries.

- An external input (EI) processes data or control information that comes from outside the boundary of the application being counted.
- An external output (EO) generates data or control information sent outside the boundary of the application being counted.
- An external inquiry (EQ) represents a combination of input (request) and output (retrieval).

The raw function point count is calculated by determining the complexity of the data or transaction function type in accordance with the number of attributes affected. Figure 2 is a summary of the how function point complexity ratings are ascertained.

Figure 2 Function Point Complexity Ratings

Input Complexity - EI	1-4 attributes	5-15 attributes	16+ attributes
0 or 1 files accessed	Low	Low	Average
2 files accessed	Low	Average	High
3 + files accessed	Average	High	High

Complexity weight : Low =3, Average = 4, High = 6

Output Complexity - EO	1-5 attributes	6-19 attributes	20+ attributes
0 or 1 files accessed	Low	Low	Average
2 or 3 files accessed	Low	Average	High
4 + files accessed	Average	High	High

Complexity weight : Low = 4, Average = 5, High = 7

File Complexity - ILF	1-19 attributes	20-50 attributes	51+ attributes
1 logical record/entity	Low	Low	Average
2-5 logical records/entities	Low	Average	High
6+ logical records/entities	Average	High	High

Complexity weight : Low = 7, Average = 10, High = 15

Interface File Complexity - EIF	1-19 attributes	20-50 attributes	51+ attributes
1 logical record/entity	Low	Low	Average
2-5 logical records/entities	Low	Average	High
6+ logical records/entities	Average	High	High

Complexity weight : Low = 5, Average = 7, High = 10

Enquiry Input Complexity - EQ	1-4 attributes	5-15 attributes	16+ attributes
0 or 1 files accessed	Low	Low	Average
2 files accessed	Low	Average	High
3 + files accessed	Average	High	High

Complexity weight : Low =3, Average = 4, High = 6

Enquiry Output Complexity - EQ	1-5 attributes	6-19 attributes	20+ attributes
0 or 1 files accessed	Low	Low	Average
2 or 3 files accessed	Low	Average	High
4 + files accessed	Average	High	High

Complexity weight : Low = 4, Average = 5, High = 7

In order to determine a final count for the system the raw count is modified by quantifying the key characteristics of the project and applying the resultant number to the raw count. These modifying characteristics are called the value adjustment factor (VAF) which indicates the general functionality provided to the user of the application. The VAF is comprised of 14 general system characteristics (GSCs) that assess the general functionality of the application. See Figure 3.

Figure 3 - Value Adjustment Factors

1.	Data communications	8.	Online update
2.	Distributed data processing	9.	Complex processing
3.	Performance	10.	Reusability
4.	Heavily used configuration	11.	Installation ease
5.	Transaction rate	12.	Operational ease
6.	Online data entry	13.	Multiple sites
7.	End-user efficiency	14.	Facilitate change

Each characteristic has six degrees of influence with associated descriptions that help determine the degree of influence of the characteristic. The degrees of influence range on a scale of zero to five as follows:

0 = not present or no influence;

1 = minor or incidental influence;

2 = moderate influence;

3 = average influence;

4 = significant influence;

5 = strong influence throughout.

The total VAF is determined by evaluating all fourteen general system characteristics and summing them to produce the total degree of influence (TDI). The TDI is inserted into the following equation to produce the value adjustment factor.

$$VAF = (TDI * 0.01) + 0.65.$$

When applied, the value adjustment factor adjusts the raw function point count +/-35 percent to produce a function point count.

The final adjusted function point count is calculated using a specific formula for development project, enhancement project, or application (system baseline) (IFPUG, 1994)

3.3 Function Point Analysis Mark II

Symons (1988) critically examined Albrectht's method and proposed a partial alternative based on overcoming perceived weaknesses. This method is based on the premise that a system consists of logical transaction types. Each transaction type being a logical input/process/output combination. In order to provide a process size measure of each transaction Symons (1988) considered the work of McCabe (1976) and Jackson (1975) to arrive at the hypothesis that a measure of processing complexity is to count the number of entities referenced by a transaction type. Referenced means any access to the entity - create, read, update or delete. It should be noted that Symons (1988) refers to entities as "anything (object, real or abstract) in the real world about which the system provides information". Symons (1988) then discusses the Mark II model in the context of using an entity relationship data model. No stipulation as to the level of normalisation, of the data model, is given. The reasoning was that the access path through an entity model involves a selection or branch or loop. Therefore the number of entities referenced by a transaction type is the measure of processing complexity. For other components of a logical transaction, input and output, the number of data element types are the measure of the size of the component. The formula for calculating Mark II Unadjusted Function Points (UFP) is:

$$UFP = N_I W_I + N_E W_E + N_O W_O$$

where

N_I = number of input data element types,

W_I = weight of an input data element type,

N_E = number of entity type references,

W_E = weight of an entity type reference,

N_O = number of output data element types,

W_O = weight of an output data element type.

It should be noted that N_I, N_E, N_O are each summed over all transaction types.

The weights were determined by calibration using data taken from twelve existing projects to arrive at the average man-hours per component. These results were then scaled to make the Mark II technique compatible with Albrecht's. This compatibility ensured all eight systems, in the calibration data set, under 500 UFP's came out to be identical on both scales. These weights were:

W_I = 0.44,

W_E = 1.67,

W_O = 0.38.

The Mark II's Value Adjustment Factor (then known as the Technical Complexity Factor) utilises the fourteen factors proposed by Albrecht (see figure 3) with the addition five new ones. These new factors are for:

1. interfacing to other applications,
2. security features,
3. direct use by third parties,
4. special user training needs,
5. documentation requirements.

The technique also allows additional factors to be used by an organisation on the provision that the factors are only those that can be derived fiom user requirements.

3.4 Feature Point Analysis

Jones (1991) developed this technique in order to "give the benefits of the function point method to real-time software, embedded software, systems software and telecommunications software". This technique was designed to overcome the perceived weaknesses of the function point technique with algorithmically complex systems. The technique uses the average complexity weighting of Albrecht's technique and adds a new parameter - algorithms with a weighting of three. In addition it reduced the weighting of the files parameter from ten to seven. The technique is summarised in figure 4.

Figure 4 - Feature Point Technique

Parameter	Complexity Weight
Algorithms	3
Inputs	4
Outputs	5
Inquiries	4
Files	7
Interface Files	7

This technique is not a simple extension to include the algorithm parameter, as alluded to by Pressman (1992), but uses a totally different method to calculate complexity. Complexity is not adjusted by using the fourteen value adjustment factors but by answering two questions that Jones (1991) claims summarises their intent. These questions relate to the problem complexity and data complexity as follows:

Problem Complexity.

1. Simple algorithms and simple calculations?
2. Majority of Simple algorithms and simple calculations?
3. Algorithms and calculations of average complexity?
4. Some difficult algorithms and calculations?
5. Many difficult algorithms and calculations?

Data Complexity.

1. Simple data with few variables and low complexity?
2. Numerous variables but simple data relationships?

3. Multiple files, fields and data interactions?
4. Very complex file structures and data interactions?

Both questions are answered and the resultant number summed together. Then a complexity multiplier is obtained from table 1 and applied to the unadjusted function point count.

Table 1 Feature Point Complexity Multipliers

Sum of Problem & Data Complexity	Complexity Multiplier
2	0.6
3	0.7
4	0.8
5	0.9
6	1.0
7	1.1
8	1.2
9	1.3
10	1.4

Jones (1991) asserts that Feature Points returns the same adjusted function point count as does Albrecht's techniques and covers the same range but in a much simpler fashion.

3.5 COCOMO

COCOMO was first described by Boehm (1981) and comprises three models which correspond to available information at different stages in the development process. Each of these models includes a number of algorithms relating product size in thousand lines of delivered source instructions (KDSI) to the development effort in months (MM_{nom}). COCOMO's three models are:

- basic COCOMO for initial estimates;
- intermediate COCOMO for when the major subsystems are determined and
- detailed COCOMO when individual modules within the subsystems have been identified.

The models' effort equations are of the form

$$MM_{nom} = a(KDSI)^{b}$$

where effort is measured in person months and size is measured in thousands of delivered source instructions (KDSI). The values of *a* and *b* depend on the model being used and the mode of development. See table 2.

These modes are Organic, Semi-detached and Embedded which represent increasingly complex software development projects.

Table 2 COCOMO coefficients

Mode	**Basic**		**Intermediate & Detailed**	
	a	b	a	b
Organic	2.4	1.05	3.2	1.05
Semi-detached	3.0	1.12	3.0	1.12
Embedded	3.6	1.20	2.8	1.20

Organic is used to describe the situation of relatively small teams developing software in a highly familiar in-house environment. Most people connected with the project have extensive experience working with related systems and the requirements and schedule are not rigorously defined. The development environment is stable with little changes to existing operational hardware and procedures.

The Semidetached mode is a mid-point between the extremes of organic and embedded. The team members have an intermediate level of experience with related systems and there is a mixture of skilled and unskilled people. The requirements and schedule are more rigorously defined than the organic mode.

The embedded mode is used for projects that need to operate with tight constraints. The resultant product must operate within a strongly coupled complex of hardware, ware, regulations and operational procedures. An embedded mode project tends to operate in new areas of application, hardware and development environments.

The coefficient values and the cost drivers described below were determined by expert opinion and a database of sixty three projects was used to refine the values.

Fifteen cost drivers are used to modify the basic equation for intermediate and detailed COCOMO by means of multipliers. These cost drivers are categorised into product, process and resource attributes. The level of each cost driver must be assessed on a six point ordinal scale. Table 3 summarises these cost drivers.

Note that all ratings categories are not applicable for each cost driver.

Table 3 COCOMO Cost Drivers

Cost Drivers	Description	Ratings					
		Very Low	Low	Nominal	High	Very High	Extra High
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	-
DATA	Data base size	-	0.94	1.00	1.08	1.16	-
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
TIME	Execution time constraint	-	-	1.00	1.11	1.30	1.66
STOR	Main storage constraint	-	-	1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility	-	0.87	1.00	1.15	1.30	-
TURN	Computer turnaround time	0.79	0.87	1.00	1.07	1.15	-
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	-
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	-
PCAP	programming capability	1.42	1.17	1.00	0.86	0.71	-
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90	-	-
LEXP	Programming language experience	1.14	1.07	1.00	0.95	-	-
MODP	Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	-
TOOL	Use of software tools	1.24	1.10	1.00	0.91	0.83	0.77
SCED	Required development schedule	1.23	1.08	1.00	1.04	1.10	-

The basic effort estimate MM_{nom} is adjusted by the product of all the cost driver multipliers.

The important points about Intermediate and Detailed COCOMO are not just the introduction of the cost drivers. Intermediate COCOMO is intended to be used when the major components of the software product have been identified. This enables effort estimates to be made on a component basis using the size and cost driver ratings appropriate for each component. The adjusted component estimates are summed to attain the total estimate. Detailed COCOMO takes the estimation process further and uses cost driver multipliers that differ for each major development phase. COCOMO also has features for handling adapted code and assessing the maintenance effort. Code re-use effects are determined by calculating an equivalent number of delivered source instructions (EDSI), and using EDSI in place of DSI in the effort equations. Maintenance effort estimates are restricted to that which is expended on the following:

- redesign and development of small portions of a product;
- design and development of small interface packages that require some redesign of the product;
- modification of the software's code, documentation or database structure.

The Basic COCOMO estimate for annual software maintenance is calculated in terms of the annual change traffic (ACT) which is the fraction of the software product's source instructions that undergo change during a year. It is calculated using the following equation:

$$MM_{AM} = ACT * MM_{nom}$$

where

MM_{AM} is the estimated annual maintenance effort;

MM_{nom} is the estimated development effort.

Boehm (1981) suggests that the annual maintenance estimate can be refined by using the Intermediate COCOMO cost drivers with the following adaptations.

- SCED is not used.
- Personnel ratings and computer turnaround are related to the maintenance staff and computer.
- New cost driver multipliers are used for RELY and MODP.

COCOMO uses a relationship between the development time (schedule) and development effort using the following equation;

$$TDEV = a(MM)^{b}$$

where

TDEV is the development time in months;

MM is the estimated effort to produce the product in man-months;

a and b are constants that depend on the mode of development as shown in table 5.The same values are used for Basic, Intermediate and Detailed COCOMO.

Table 4 COCOMO Schedule Equation Coefficients

Mode	a	b
Organic	2.5	0.38
Semi-detached	2.5	0.35
Embedded	2.5	0.32

The COCOMO model also defines details such as a man month consists of 152 hours of working time and perhaps most importantly provides a phase and Work Breakdown Structure (WBS) for which the model applies. Boehm (1981) also details assumptions such as the project "enjoys good management" and "the requirements specification is not substantially changed after the requirements phase". Boehm's work is thorough and demonstrates an excellent understanding of the realities of software development.

Boehm (1987) also developed an improved version of COCOMO which is based on a more modern process model which includes risk management and can be used to predict the costs of Ada projects.

3.6 COCOMO 2.0

COCOMO 2.0 is currently under development and as yet there are only unpublished preliminary manuals available. This work will be very important and impact on all future software estimating models. It was recognised that COCOMO had increasing difficulty in estimating the costs and schedules of business software, object oriented software, software developed using an evolutionary approach and software that is a composite of commercial packages.
COCOMO 2.0's construction has been guided by an anticipated model of future software development practices. This model's components are outlined below.

- **End-user programming** - where applications will be developed using application generator tools such as spreadsheets, query systems and parameter driven specialised systems.

- **Infrastructure** - where applications will be in the areas of operating systems, data-base management systems and networks operating systems together with the user interface tools.
- **Application Generators** - where the bulk of the tools used by the end users will be developed such as financial analysis tools, project management tools, etc.
- **Application Composition** - where applications too complex for a single tool will be created from several inter-operable components.
- **Systems Integration** - where large scale, embedded or unusual systems will be developed that require a significant amount of customised software development.

COCOMO 2.0 provides a suite of increasingly detailed estimation models in order to satisfy the different practices. The end user practice is not seen to need a COCOMO 2.0 model as the applications are simple and will be developed in a small number of days. The first model addresses the Application Composition practice which comprises applications that cannot be built using a specific tool such as a spreadsheet. However the application can be created using a number of diverse packages. The approach used is called Object Point estimation. This technique is similar to Function Point analysis in that it uses a like process which is outlined below.

1. Assess object counts: estimate the number of screens, reports and 3GL components that comprise the application.

2. Classify each object instance into simple, medium and difficult complexity levels using supplied tables.
3. Assign a weight to each instance using a supplied table.
4. Add all the object instances to obtain an Object Point count.
5. Estimate the percentage of re-use expected to be achieved in the project using the following formula:

$$\text{New Object Points} = \frac{(\text{Object Points}) * (100 - \%\ \text{Re-use})}{100}$$

6. Determine a productivity rate (productivity being measured in terms of the New Object Points per person month) from the supplied table.
7. Compute the estimated person months.

The second and more detailed model, Early Design, uses unadjusted Function Points as a sizing metric. The VAFs are not used as COCOMO (1995) advises that the characteristics and relative weighting are inconsistent with their experience. The unadjusted Function Points are translated into source lines of code (SLOC) and then KSLOC by using tables such as those provided by Jones (1991). A set of cost drivers is then applied.

The third model, Post Architecture uses KSLOC as per the Early Design model but uses a more comprehensive suite of cost drivers.

3.7 Expert Judgement

The techniques in this area involve consulting with experts to obtain their opinion and consequent estimate as to the effort cost and schedule factors for a particular project. An expert can factor in elements of a project such as the skill of the people involved, the similarity with past projects and political aspects of the development. If a single expert's opinion is obtained then the result can be subject to bias and an unfamiliarity with major aspects of the system.

To overcome the difficulties associated with a single expert an number of group consensus techniques have evolved such as the Delphi technique. This technique originated at the RAND Corporation and the Wideband Delphi version is described by Boehm (1981).

The use of the Wideband Delphi technique proceeds as follows.

1. A coordinator provides each expert with a specification of the system and an estimation form.
2. A group meeting is held in which the project and estimation issues are discussed.
3. The experts form an estimate individually and anonymously including rationale they feel may be required.
4. The coordinator summarises all the estimates and distributes to all the experts without the rationale.

5. Another group meeting is held which focuses on the areas where there is a wide divergence of opinion. These areas are discussed in depth to ensure all experts have an understanding of the issues involved.
6. Another estimate is made by the experts individually and anonymously and steps 4 to 6 are iterated to obtain convergence.

This method ensures that there is good understanding of all the issues involved through communicating at the meetings whilst also minimising the impact of any dominant individual.

This technique has been extended by Hope (1993) whereby detailed estimating forms (see attachment 1) are provided to the experts that require them to make optimistic, probable and pessimistic estimates of both cost and effort. The elements of the forms were derived from analysis of five large projects implemented on a national basis within Telecom Australia. The method has not been validated however proved useful to identify cost and effort factors not considered by other known techniques. For instance in one project with a total cost of $4.8m, $1.3m was identified to environmental costs (Telecom, 1992).

A formula

$$Estimate = \frac{Optimistic_{Tot} + (4^{*} Probable)_{Tot} + Pessimistic_{Tot}}{6 * E_{Tot}}$$

is used to give a weighting to the sum of the estimates. E_{Tot} is the number of experts providing estimates. The rationale behind the equation is the standard deviation of a beta distribution.

3.8 Other Techniques

There are numerous other estimating models available. These are listed below, however are not described as they add little more to this research. These other techniques are:

- TRW Wolverton Model
- TSDC Model
- Walston-Felix
- SOFTCOST
- PRICE SP
- ESTIMACS
- Bailey-Basili Meta Model
- Putnam's model
- Parr
- Jensen
- COPMO

4. Estimating Technique Survey Analysis

Estimating technique utilisation which was obtained from three published surveys, one conducted in the USA another in New Zealand and the third in the Netherlands. Wydenbach & Paynter, (1995) also reported Heemstra & Kusters' (1989) results from a similar survey conducted in the Netherlands. (Hihn & Habib-agahi, 1991: Wydenbach & Paynter, 1995).

Hihn & Habib-agahi's (1991) survey contained four categories which were informal analogy, formal analogy, rules of thumb and models. Their research was limited to the technical divisions of a single organisation, the Jet Propulsion Laboratory. The categorisation was not rigorous with overlaps and the data "reflects the authors' interpretation of what techniques were the dominant ones".

Wydenbach & Paynter, (1995) contained eight categories and their survey was conducted by mail on New Zealand organisations involved in software development. The data indicates that whilst eighty percent of respondents consider the estimation process to be important and ninety eight percent make some form of estimate only twenty five percent use a formal approach. The most common formal estimation method was found to be function point analysis. Table 5 below is a summary of data contained in these surveys. Where a method was not considered in a survey it has been marked not applicable (N/A).

Table 5 Percentage Comparison of Estimating Techniques Used.

Estimation Methods Wydenbach & Paynter	% of total respondents (209)	Estimation Methods Heemstra & Kusters	% of total respondents (369)	Estimation Methods Hihn & Habib-agahi	Respondents (83)	
					Primary %	Secondary %
Expert judgement	86%	Consult an expert	26%	Rules of thumb (expert)	6%	55%
N/A	-	Intuition	62%	Analogy, informal	83%	34%
Reasoning by analogy	65%	Analogy method	61%	Analogy, formal	4%	0%
Bottom-up	51%	N/A	-	N/A	-	-
Models	26%	Parametric models	14%	Models		
Price-to-win	16%	Price-to-win	8%	N/A	-	-
Top-down	13%	N/A	-	N/A	-	-
Available capacity	11%	Capacity problem	21%	N/A	-	-
Other	0%	Other	9%	N/A	-	-

Heemstra & Kusters' (1989) data indicates that only fourteen percent use a formal model approach. This difference from the New Zealand survey (26%) was explained by Heemstra & Kusters' (1989) large percentage of the "other" category purports to contain non-commercial models.

It is interesting to note that in all surveys conducted above, the largest category was estimating by analogy.

Park et al (1994) conducted a survey in 1993 to assess the need for improvements in software cost estimating and as an input to the prioritisation of the work at the Software Engineering Institute at Carnegie Mellon University. The survey was basic with only eight questions, one of which was contact information. They distributed the survey widely to groups affiliated with the SEI and those who have an obvious interest in software estimating such as the COCOMO user group. This has, no doubt biased any results obtained. It is also of interest to note they only received 249 responses. The question of most interest in this research was "What improvements would be of most help?". This question did not have a structured reply and the authors grouped according to the general areas they addressed and advised "... everyone sees a need to improve software estimating, but few see the same needs". The general area groupings used were size, models, databases, metrics and process. Unfortunately Park (1994) did not supply the total data, however, gave forty nine examples of the responses. Of these, fourteen were concerned with the improvement of the sizing of a software project and thirty one advised a standard model and/or process with which to develop and record estimates would be of benefit.

It is unfortunate that a comprehensive survey that addresses and analyses the needs of this research was not found. Work is in progress at Edith Cowan University to address this gap.

It can be concluded from these surveys that the more formal and structured estimating techniques like COCOMO are not widely used in practice. The majority of software practitioners appear to estimate by using expert judgement and analogy.

5. Theoretical Framework

The estimating of software projects has important ramifications on organisations who are making decisions based on the estimates and on the teams and personnel who undertake the projects. Therefore it is important that any measures derived for estimating purposes must be based in measurement theory if they are to have any mathematical validity. It is apparent that a number of "metrics" in the Software Engineering paradigm fail to take heed of the available theory and hence the metrics espoused are flawed (Fenton 1994).

Measurement is defined by Fenton as "the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules". An entity can be either an object, such as a requirements specification, or process of interest, such as the requirements phase of a project. An attribute is a property of an entity such as the length of a requirements document. There are two types of measurement, direct and indirect. Direct measurement is where the measurement of an attribute does not depend on the measurement of any other attribute. Indirect measurement is an attribute that comprises the measurement of one or more other attributes.

Hence it is important to note that measurement is a *defined mapping* of numbers or symbols to an attribute which must preserve any intuitive or empirical observations about the attribute.

For instance, we could measure the length of a requirements document by mapping to the attribute *length* the number of pages or the number of words comprising the document. To be clear about the attribute we would have to have a formal definition or model for the requirements document that defined the rules under which the measurement took place in order that the length could be stated unambiguously and in a repeatable fashion. For instance a requirements document model would have to cater for various aspects that could impact on the attribute such as page size, font size, line spacing, standard contents, etc. It is interesting to note where common measurements are taken this definition applies. For instance, in the measurement of the height of a person rules apply as to the person's attitude, ie standing with feet on the ground and the disposition of the footwear before mapping the person's length to a number system.

Fenton (1991) is of the opinion that where no previous measurement has been performed or the attributes are not well understood one should attempt to obtain direct measures in order to gain an understanding of the entity and attributes in question.

For measures to be valid it is generally considered that they should obey the representation condition of measurement theory (Fenton, 1991). The representation theory of measurement has a mathematical framework based on sets, relations, axioms and functions. The components are :-

- Empirical relation systems which determine the axioms that characterise any empirical observations or relations between the entities. The set of entities E, together with the set of empirical relations R, is called an empirical relation system (E,R) for the attribute under observation. For example, the attribute length of a document leads a binary relation "is longer than" and this satisfies the axiom of transitivity. That is, if document A is longer than document B which in turn is longer than document C, then we may infer that A is longer than C. Relations do not have to always be binary, for instance, "is long" would only apply to an single instance of a document.

- The representation condition is required for measurement in order that the attribute defined in the empirical relation system (E,R) can have a mapping M into a numerical relation system (N, P) in such a way that all empirical relations are preserved. That is M maps attributes in E to numbers in N and empirical relations in R are mapped to numerical relations in P. Note the representation condition asserts that the correspondence between empirical and numerical relations is two way. For instance with the document example above if we considered E as the set of all documents and R contains the relation "longer than". Then a measure M of length would map E to the set of positive integers and "longer than" to the relation "> ". The representation condition asserts that document A is longer that document B, if and only if $M(A) > M(B)$.

It should be noted that empirical relations are normally established by subjective means as a precursor to more objective forms.

- The scale types that can be meaningfully applied to the measurement of an attribute are dependant on the representation mapping ***M*** from an empirical relation system ***E*** to some numerical relation system ***N***. If such a representation exists then the triple (***E***,***N***, ***M***) is called the scale.

A framework for the validation of software measurement has been proposed by Kitchenham et al (1995) which should prove useful in this work. The framework is based on Fenton's work and has the goals of helping both the areas of research and practice by facilitating the understanding of :

- measure validation
- validation work assessment
- appropriateness of measures in a given situation.

6. Analysis of Existing Models

6.1 Function Point Analysis

There is confusion as to what function points are actually measuring. Albrecht's Function Point Analysis and Jones' Feature Point Analysis are assumed to either measure size or functionality as perceived by the user of the software product. The view held by the International Function Point Users Group, IFPUG (1994) is somewhat confusing as they discuss both "as a measure of the functional **size** of information systems" and a "measure of functionality that the user requests and receives". Albrecht (reported in Symons 1988) stated that the "measure isolates the intrinsic **size** of the system from environmental factors...".
However, function points are calculated from the sum of a number of different elements and therefore appear to be an attribute in their own right derived from an attribute relationship model. As Kitchenham et al (1995) espouses, "the term function point does not seem appropriate; function points might be better renamed as functionality or user requirement size".

However, more fundamental issues need to be addressed with function points. Function points are the sum of five elements derived from the number of inputs, outputs, inquiries, data and interface files. The input element is based on the number of data elements involved in each system input - see figure 2 for details. If the number of data elements involved in all inputs were summed then this would be an acceptable measure of input data size. However, the function point model involves classifying each input as simple, average or complex, using an ordinal scale, according to the number of data elements and files accessed. The values derived are then mapped to numbers and summed. It would appear that the function point model is in violation of basic measurement theory in that you cannot sum ordinal scale measures. Also the counting rules mean that the smallest system has a value of three which implies that the values are discontinuous and there is no unit value. This is another violation of the measurement framework. These arguments are also applicable to Feature Points.

Albrecht's Function Points have also been criticised by Symons (1988) on a number of grounds. These being:

- It is difficult to define the basic counts objectively.
- The complex, average and simple classification is over simplified.
- The choice of weights for the initial classification and calculation of the technical complexity factor was determined subjectively and based on experiences at IBM.
- Internal complexity is treated twice, during the initial classification and during the calculation of the technical complexity factor.
- The effect on function point counts of comparing a group of independent systems linked by interfaces and a single fully integrated system is counter intuitive.

There are also problems with the value adjustment factors in several ways. Jeffrey, Low & Barnes (1993) has shown that the complexity adjustments do not improve effort predictions and there was no significant differences between unadjusted and adjusted function points as effort predictors. Kitchenham & Kansala (1993) have reported similar results.

Fenton (1994) is of the opinion that using the VAF adjustment, for a model that measures system functionality, is "analogous to redefining measures of height of people in such a way that the measures correlate more closely with intelligence'. Other concerns with VAFs is that they are open to interpretation and it is easy to see overlap. See table 6 for details of overlap.

Table 6 VAF Overlap

	VAF	VAF Overlap
1.	Data communications	6, 8, 2
2.	Distributed data processing	1
3.	Performance	6, 8
4.	Heavily used configuration	
5.	Transaction rate	
6.	Online data entry	1, 3, 8
7.	End-user efficiency	6, 8
8.	Online update	1, 3, 6, 7, 14
9.	Complex processing	
10.	Reusability	
11.	Installation ease	
12.	Operational ease	
13.	Multiple sites	
14.	Facilitate change	

Therefore the use of VAFs are subjective and depends on interpretation as to what the person conducting the count perceives a sbeing in each category. VAFs were formulated in 1984 and as such are not wholly relevant to modern software products and development environments. For instance, the graphical capabilities required and the provision of inquiries as defaults in fourth generation languages are not easily accounted for. One of the more important modifiers to most other estimating techniques are aspects of the quality of the software product, most of the quality attributes are missing from the function point model. The application of the model will always give a linear result which is counter-intuitive in that the amount of work increases geometrically as the size of the project increases ie large projects take a significant amount of more work than small ones.

The applicable scope of a software project covered by function points is undefined. This would appear to be a major omission as one of the stated aims of IFPUG (1994) is to provide a normalisation factor for software comparison. The least the Function Point models should do is outline the lifecycle phases and major activities that are part of the "size".

Mark II function points take a different approach in that the function points are derived from the inputs, outputs and entities for each business transaction. The transaction input size is the sum of the data elements that are input into the system; the transaction output size is the sum of the data elements that are output from the system; the transaction data processing size is the sum of the number of entities referenced when the transaction is processed. These values are summed for each transaction and therefore represent three different size attribute elements that are input into the system. The model requires that the attribute values be weighed and summed. The weights are different for each attribute and represent the development

effort involved. This violates the measurement framework if we regard Mark II function points as a size or functionality measure, however, it could be considered to be an effort measure as the weights are derived from the number of manhours involved in delivering each component.
It must be concluded that there are major problems associated with the meaning and construction of function point measures. It is interesting to note that there is little work published on the validity of the measures as to their predictive capability.

From the project data the initial size of one project was estimated at 1477 function points and although a count was never conducted on the final product it was estimated the final system was in excess of 3500 function points. This is based on an extrapolation from the forty one entities of the data model used in the initial estimate to the final having one hundred and twenty three entities (Telecom 1993). Whereas is another project, Telecom (1992), the initial count was 1230 function points and the count on the delivered system was 1876 function points. All these counts were conducted in the same environment by the same people using the same delivery systems and count mechanism. From this example it can be seen that function point counting can be inconsistent and subject to a great deal of variation. Unfortunately no published material could be found that compared actual function point counts with estimated ones.

6.2 COCOMO & Lines of Code Measures

The COCOMO model depends on estimates of KDSI (thousands of delivered source instructions) for its major input which is not really measurable until the software product has been implemented. As such this measure is subjective although estimates should become more accurate as the project progresses. Therefore it would seem that a difficult prediction problem, effort, is being replaced with an equally difficult prediction problem - size. Also the COCOMO models require that the modes of development (organic, semi-detached or embedded) be determined and in the Intermediate and Advanced models fifteen cost drivers must also be rated. Therefore the objectivity of the inputs to the COCOMO models are questionable.

The use of KDSI has other problems which are as follows.

- As Jones (1991) states there is no industry standard definition for a line of code (LOC).
- Some languages such as Pascal and Ada allow many logical statements per physical line whereas other languages such as COBOL have physical line requirements.
- The types of lines that are counted need to be defined as most procedural languages include four different kinds of source statements executable lines, data definitions, comments and blank lines. Data definitions can also cause problems as *n* variables can be declared in one statement or *n* statements for the same logical outcome.

- The concept of a LOC is not represented in some fourth generation languages such as Oracle Forms. These languages also tend to use third generation type languages in part, thereby compounding the problem.

The COCOMO models are extremely comprehensive and, being based on well documented empirical studies, tend to be intuitively sound.

6.3 Conclusion

Function points do not relate to any lifecycle model or any set of activities. Therefore in addition to the problems mentioned above it is difficult to know what activities can be included when determining productivity and costing factors. That is, is it allowable to include such elements as the effort to produce systems manuals, the cost of development tools etc in the production of the system under investigation.

COCOMO has a model on which it is based and only covers the software lifecycle from requirements to implementation for those activities in the work breakdown structure nominated. However, it has all the problems espoused above and especially those associated with lines of code measures.

It should be noted that no published material was found relating to experiences with the Wide-Band Delphi method.

7. Proposed Model

7.1 General

The proposed model outlined in this section cannot be considered complete, however, has an underlying principle of providing an estimate for a **total software project**. That is all costing and effort elements required to deliver system are considered. A **TOTAL** project estimate is required as only this will provide the information and costing that will allow management to make valid decisions on the viability and feasibility of the proposed system.

Estimation components of a software project consist of the product and the process that produces it. However in order to compare different projects there must be agreement as to the elements that will be counted as part of the cost of the software projects in question. As related earlier a project with a total cost of $4.8m had $1.3m attributed to environmental costs (Telecom, 1992). On examination these costs related to changes and provision of both electrical and network cabling, provision of lighting that reduced screen reflections and the provision of office furniture that was ergonomically sound. Therefore this organisation considered it to be reasonable to associate these costs to a single project. Other organisations may have considered these as infrastructure costs and handled them in a different manner. If another organisation did not consider these environmental costs then any comparison between projects would be flawed if the information was not normalised in some manner to allow project comparison. Whilst this example is somewhat obvious and easily catered for, other costs are not so easily recognised and catered for in the data collection. For instance in another project twenty three percent of the total number of hours on the project could be attributed to unpaid overtime (Telecom 1993). Only costing the hours worked and paid would give an unrealistic view of the productivity factors that could be used in future projects.

Therefore elements of the total project need to be defined and those elements that are particular to a single project extracted before comparisons are made between projects. Therefore the ideal estimating model for a project would be to add all known factors (F) together as follows:

$$\text{Estimate} = F_1 + F_2 + F_3 + \dots + F_i$$

Each factor could have a different effect on the project and hence a multiplier (M), for each factor, would be appropriate which leads to:

$$\text{Estimate} = M_1F_1 + M_2F_2 + M_3F_3 + \dots + M_iF_i$$

However, it is known from various studies such as Boehm (1981) that some factors have a non-linear (NL) effect on the project (eg size) and therefore the equation would be of the form:

$$\text{Estimate} = (M_1F_1)^{NL1} + (M_2F_2)^{NL2} + (M3F_3)^{NL3} + \dots + (M_iF_i)^{NLi}$$

However, due the immaturity of software estimating and the wide variance in results reported from empirical studies some factors would not be relevant to consider as their impact would be within the scope of the variance. This leads to single factor models such as COCOMO ($MM_{nom} = a(KDSI)^{b}$) whose result is modified by the application of fifteen cost drivers.

It would appear that these types of estimating models are valid for the environment in which they were derived and are useful as long as that total environment remains stable. This is evidenced by COCOMO (1995) where it is advised that COCOMO and Ada COCOMO were reasonably well matched to the large customised projects from which they were modelled however are not suitable for future environments.

7.2 The Model

A model is required that considers all the factors involved in the construction of a software product. This is required as different classes of projects will contain different components and be affected in different ways by the environment in which they are produced. Some elements of such a model are contained in figure 5.

The result of such a model would be an estimating handbook for software projects perhaps in a similar fashion to the estimating handbooks used by architects and builders. This handbook would contain all the elements that could constitute a project estimate and the various factors that affect each element. The handbook would have to continue to evolve as environments changed and data was collected to improve the model. A candidate list is contained in Attachment 1.

Figure 5 Some Elements Impacting on a Project Estimate

Product	Size	Data amount and complexity; Processing amount (functions) and complexity;
	Target Environment	Mainframe based; PC based; Distributed client server; Available memory and processors; Network traffic intensity; Combinations of the above;
	Lifecycle Scope	What phases are included (esp maintenance) Project elements Support hardware and software; System hardware and software Users time; User training; Data take up and validation;
Quality	Attributes	Reliability; Maintainability etc Some modified form of QFD may be applicable.
Process	Politics	How acceptable is the system to the users; User commitment; Does it fit into the organisation's strategy; Management commitment
	Developer's attributes	Management capability; Personnel capability - skills, experience in the tools platform and application domain; Availability and continuity;
	Risk	Relates to product and process
	Development environment	Hardware; Software tools; Management systems - QM, PM; CM; ... Multi-site development
	Constraints	Schedule Building for re-use; New techniques and tools being utilised - Hawthorne effect - results may not translate to normal practice;

The following discusses various aspects of the model.

7.2.1 Product

7.2.1.1 Size

Obtaining size estimates that are reliable is difficult and subject to a wide range of uncertainty. As Boehm (1981) observed "the biggest problem in today's algorithmic software cost models is the problem of providing sound sizing estimates". From the research it would appear the models utilising function points and lines of code still have major problems today. Verner and Tate (1992) reported in a United States Air Force experiment which compared six software size estimation models the results ranged from 6622 to 36700 lines of code. The actual size was 9177 lines of code. Object points have been mooted as an answer, however, more research needs to take place in order to validate or refute them.

Data Size - It would appear that for data a case could be made to count the attributes/fields/data elements that a user can see. This would give a measure on a ratio scale ie we have a zero point. Then these could be formed into a data model in third normal form and the number totalled. A non-linear function would be required, for as the total increased, it could be assumed that the inter-relationship between the entities and hence the complexity of the application would increase which would lead to greater effort and cost. Brooks (1975) and Jones (1991) provide adequate evidence on the non-linear effect of size on a projects cost, effort and duration.

Processing Size - One method would be a simple count of the functions to be provided. There is a need for a non-linear expression to designate the complexity of each function as this will impact on the overall estimate. Estimating lines of code has the problems discussed previously. Also some lines of code are more complex than others and hence require a greater intellectual effort to produce. For example if you had a recursive routine that called another recursive routine then the effort in writing, testing and de-bugging would be more that that involved in two routines that formatted a simple output.

Verner and Tate (1992) support the notion of a generic sizing model that is not fixed but the partitioning can depend on the development technology. This model follows a bottom-up approach that identifies the components of a system and allows different estimation equations for different component types.

7.2.1.2 Target Environment

The target platform(s) will not only have a effect on the development cost but also on the ongoing maintenance. This will be evidenced mainly in the configuration management costs. For example in Telecom (1993) an application was implemented in a client server environment and distributed across Australia with major regional clients in the capital cities. This involved areas of work in data communications analysis and installation, implementation planning and execution, configuration management etc. In developing a single PC based application these items would not be relevant.

7.2.1.3 Lifecycle

The work breakdown structure for a project needs to be defined. All activities, effort and cost elements need to be defined in order that projects can be compared and an historical information recorded. A definitive method for recording items, such as man-hours, also needs to be established. It is interesting to note in the research conducted it was found that only Boehm (1981) defined this element in the COCOMO model. The International Standards Organisation ISO/IEC (1995) has published a comprehensive document detailing lifecycle processes that would form an internationally recognised and publicly available source for this estimating element. Project Elements such as support hardware and software user training, user procedures and policy changes, environmental costs etc could also be incorporated into the Work Breakdown Structure.

7.2.2 Quality

Most models incorporate some of the quality elements into their models such as COCOMO's reliability cost driver, however most leave the majority of the recognised quality attributes out. See figure 6 for software quality attributes. Weinberg (1971) proved the goal set for a programming team was usually the one achieved. His experiments, using five programming teams, also provided evidence that given the goal of usability or maintainability the cost of development was higher than it would have otherwise been. All the quality attributes of a system should be considered and a modified form of quality function deployment applied as partially devised by Thomsett (1993). Thomsett (1993) requires all project stakeholders to rate the quality attributes on a scale -3 to +3 with 0 being the nominal quality provided in a system. This quality model would require empirical experimentation and calibration to make it useful. However, even without this rigour it is still a useful approach as the quality cost drivers for a project are explicitly stated.

Figure 6 Software Quality Attributes

• Correctness	Does it accurately do what is intended ?
• Reliability	Does it do it right every time ?
• Durability	Will it continue to work after a part fails ?
• Efficiency	Does it run as well as it could ?
• Generality	Does it cover the whole problem domain ?
• Integrity	Can it be trusted to handle unusual conditions for which it was not explicitly designed ?
• Useability	Is it easy to use ?
• Readability	Are its processes easily understood ?
• Testability	Is it easy to check and verify correct ?
• Maintainability	Is it easy to fix ?
• Flexibility	Is it easy to adapt and extend ?
• Portability	Can it be easily converted ?
• Compatibility	Does it interface well with other systems ?
• Security	Is it safe from unauthorised modification or use ?

7.2.3 Process

The process of developing software is complex and involves numerous processes that are all interrelated. This is another reason for the difficulty in estimating and managing software projects. To arrive at an estimate that will predict the outcomes accurately not only do all the elements constituting the development have to be known but also their interrelationships and effects they have on the dynamics of the system being estimated. Figure 7 from Abdel-Hamid and Madnick (1989) shows such a model. Obviously some automated tool is required when analysing such models. Other elements that are of note are discussed below.

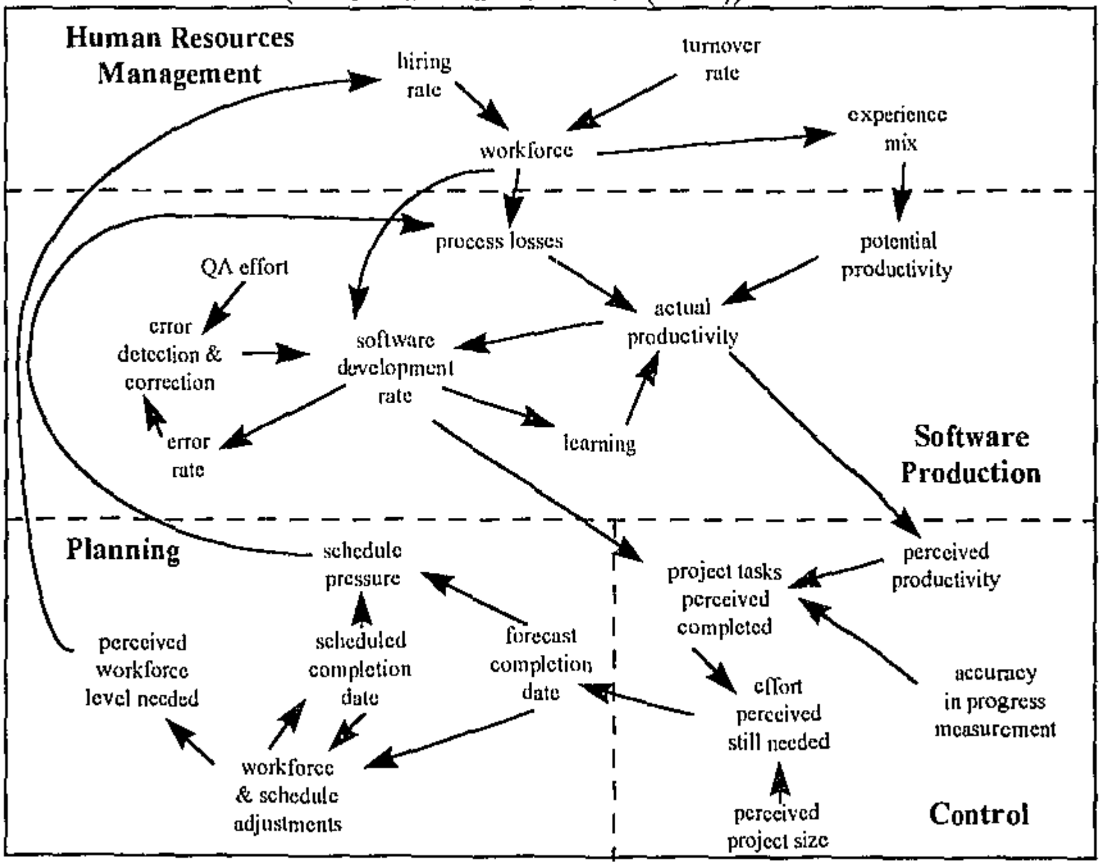
7.2.3.1 Politics

The management of organisational politics is of great importance and if it is not done well can have a detrimental affect on the project. The management of all the stakeholders is essential. Thomsett (1993) discusses the management and categorisation of stakeholders in order that the project team focuses on the most critical areas. In the project described in

Telecom (1992) little explicit attention was made in this regard however in Telecom (1993) budgeted items amounting to $60K were allowed. This enabled the system to be more readily accepted and ensured there were designated people in each state who would "champion" the system. COCOMO 2.0 also includes this stakeholder management as part of the TEAM rating components.

Figure 7 Systems Dynamic Model

(Abdel-Hamid and Madnick (1989))


Human Resources Management
hiring rate
turnover rate
workforce
experience mix
process losses
potential productivity
QA effort
actual productivity
error detection & correction
software development rate
learning
error rate
Software Production
Planning
schedule pressure
project tasks perceived completed
perceived productivity
perceived workforce level needed
scheduled completion date
forecast completion date
effort perceived still needed
accuracy in progress measurement
workforce & schedule adjustments
perceived project size
Control

7.2.3.2 Developer's Attributes

Boehm (1981) places the attributes of the developers as the element that has most impact on the estimate for a project. This is also recognised in COCOMO (1995) where the same level of importance is attached. Various studies such as in Brooks (1975) and Weinberg (1971) have shown that there is a vast difference in productivity between development personnel. The differences can be on the order of twenty to one. The differences can also vary from development task to development task. Modelling and measuring the skills of personnel is a difficult task that changes over time and is also dependant on the environment in which a particular person is operating in. One method would be to have nominal delivery rates for the activities defined in the WBS and modify these based on individual's performance data.

7.2.3.3 Risk

Software Risk Management is an emerging discipline whose objectives are to identify, analyse, address and mitigate software risk items before they become threats to the software products and systems. As has been alluded to previously the outcome of software development activities are probabilistic. Software risk management applies techniques for determining probabilities and increasing the chances of success. Another effect of this risk management process is the reduction of re-work. The direct impact on a project estimate would be the cost of risk management which consists of assessment, analysis, mitigation and tracking. (Boehm 1992: Charette 1989).

7.2.3.4 Development Environment

DeMarco and Lister (1987) conducted experiments that showed the development environment had a major affect on the productivity of software development personnel. They showed, in their experiments that if one person in an organisation performed well then so did others. DeMarco and Lister (1987) said "... the best organisation worked 11.1 times faster that the worst organisation". This they attributed, in the main, to the workplace with the control of noise and provision of adequate work space having major productivity affects. Software development is essentially an intellectual activity and constant interruption or distracting noise makes it difficult for competent people to work effectively.

The management systems within the organisation will also impact upon the productivity. This is closely aligned to the processes that are being undertaken. COCOMO 2.0 addresses this area explicitly with reference to the Capability Maturity Model of Carnegie Mellon University for the determination of their process maturity cost driver.. A software quality management system has as one of its goals the reduction of re-work. Organisations that allow errors to propagate throughout the development have lower overall productivity.

Other aspects such as the development environment stability and availability, team distribution (collocated or dispersed), tool sophistication, etc. All would need to be detailed and the project effect determined from historical data.

7.2.3.5 Constraints

Various constraints can be placed on a project the chief one being any schedule that is tighter than that initially estimated. Schedule constraints if applied have a disproportionate affect on manpower requirements. Brooks (1975) was one of the first to make this point in that "the man month as a unit for measuring the size of job is a dangerous and deceptive myth" because it implies that people and effort are interchangeable. They are only interchangeable if there is no communication between the people involved. In software development communication and interrelationships between activities and people is high. As Brooks (1975) says "if each part of a task must be coordinated with each other part of an activity, the effort increases $n(n-1)/2$". Therefore three people require three times as much intercommunication as two and four six times as much as two etc. This can lead to the effort in communication outweighing any benefit of task division.

Other management imposed constraints may also impact upon the estimate. For instance if a proportion of the system has to be developed for re-use then greater effort is required in ensuring the components are sufficiently generic to be re-used. The same applies to the use of new tools and techniques as there will be a learning curve involved.

7.3 Summary

As can be seen from the above there are numerous factors involved in estimating a software project. These factors range from consideration of development hardware to the skills of individuals involved in project activities. Not all will be relevant to all projects, however, all need to be considered as the potential to impact on the project estimate can be great. As stated earlier this is only a framework from which an estimating technique can be developed.

8. Conclusion

Estimating the size, effort, duration and cost of a software project is an essential aspect of Software Engineering as these are the fundamental drivers for all project decisions. This research has investigated and analysed the major software project estimating techniques in use today. As can be seen there are significant weaknesses with the existing models and techniques for estimating software projects. These range from not catering for modern development environments (4GLs, object oriented techniques and languages) to those that are theoretically unsound and not based firmly in measurement theory. It would also appear that most methods are too simplistic and fail to adequately deal with all the complexities involved in developing a software product. This would appear to be a inherent attribute of the software industry where a "**silver bullet**" is always being sought.

The research has also revealed, through the analysis of existing surveys, that these techniques are not widely used and most practitioners use expert judgement or analogy to determine project cost and effort. This is despite most techniques being available for ten to fifteen years now.

The proposed model is only a framework and more work is required to quantify it and to determine how it could be tailored to suit an organisation. The complexity and dynamics of the software development process and the confounding organisational factors make it, except in the most general terms, very difficult to compare between organisations. Any comparison between software projects across organisations would have to be normalised. That is a standard of not only the activities and cost elements involved but also the data collection and definition mechanisms would also have to be agreed.

Estimating without either a detailed requirements document or design document is a problem as this is the first time the data and functions required by the system are expressed in a detailed form. Perhaps a change in terminology is required and that all efforts to predict the size, cost, effort and duration prior to these documents being available should be referred to as **forecasts**.

A builder of houses uses an estimating workbook that spans several pages, however, in the software industry we appear to seek a simple technique with a few parameters on one page to estimate products that are orders of magnitude more complex to build than a house. This research has revealed that an estimating framework that considers all the parameters of a project in detail is not inappropriate.

9. References

Abdel-Hamid, T. K., Madnick, S.E. (1989). Lessons Learned From Modelling the Dynamics of Software Development. Vol 32, No 12 Communications of the ACM.

Abdel-Hamid, T. K. (1993). Adapting, Correcting, and Perfecting Software Estimates: A Maintenance Metaphor. Vol 26, No 3 Computer, IEEE Computer Society.

Albrecht, A.J. (1979). Measuring Application Development Productivity. Proceedings - Joint Share/Guide IBM Application Development Symposium pp 83-92.

Basili, V.R., Rombach, D. (1988). The TAME Project. Towards Improvement-oriented Software Environments. Vol 14, No 6 IEEE Transactions on Software Engineering, IEEE Computer Society.

Boehm, B.W. (1981). Software Engineering Economics. New York: Prentice Hall.

Boehm, B.W. (1984). Software Engineering Economics. Vol 10, No 1 IEEE Transactions on Software Engineering, IEEE Computer Society.

Boehm, B.W. (1987). Ada COCOMO: TRW IOC version.. Third COCOMO User's Group Meeting.

Boehm, B.W. (1992), Risk Control, American Programmer Vol 5 No 7, pp 2-9, New York, NY.

Brooks, F.P. (1975). The Mythical Man Month. Addison-Wesley.

Charette, R.N. (1989). Software Engineering Risk Analysis and Management, McGraw-Hill Book Company, New York, NY.

COCOMO (1995). COCOMO 2.0 Model User's Manual - Version 1.1. University of Southern California.

Daskalantonakis, M. K. (1992). A Practical View of Software Measurement and Implementation Experiences within Motorola. Vol 18, No 11 IEEE Transactions on Software Engineering, IEEE Computer Society.

DeMarco, T. Lister, T. (1987). Peopleware: Productive Projects and Teams. Dorset House.

Dreger, J. (1989). Function Point Analysis. Englewood Cliffs, NJ: Prentice Hall.

Fenton, N. (1994). Software Measurement: A Necessary Scientific Basis. Vol 20, No 3 IEEE Transactions on Software Engineering, IEEE Computer Society.

Fenton, N. E. (1991). Software Metrics - A Rigorous Approach. London: Chapman & Hall.

Garmus, D., Herron, D., (1996). Measuring the Software Process: A Practical Guide to Functional Measurements Englewood Cliffs, NJ: Prentice Hall.

Gilb, T. (1988). Principles of Software Engineering Management. Wokingham: Addison-Wesley.

Hihn, J., Habib-agahi, H. (1991). Cost Estimation of Software Intensive Projects: A Survey of Current Practices. 13th International Conference on Software Engineering, IEEE Computer Society.

Hope, S. (1993). Software Estimating Workshop Course Notes. Spiral Technology Pty Ltd.

Humphrey, W.S. (1989). Managing the Software Process. Reading, Massachusetts: Addison-Wesley.

Humphrey, W.S. (1995). A Discipline for Software Engineering. Reading, Massachusetts: Addison-Wesley.

IFPUG, (1994). Function Point Counting Practices Manual, Release 4.0. International Function Point Users Group, Westerville: Ohio.

ISO/IEC 12207:1995. Information Technology - Software Lifecycle Processes.

Jackson, M. (1975). Principles of program Design. London, Academic.

Jeffery, D. R., Low, G.C., Barnes, M. (1993). A Comparison of Function Point Counting Techniques. Vol 19, No 5 IEEE Transactions on Software Engineering, IEEE Computer Society.

Jones, C. (1991). Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill.

Jorgensen, M. (1995). Experience With the Accuracy of Software Maintenance Task Effort Prediction Models. Vol 21, No 8 IEEE Transactions on Software Engineering, IEEE Computer Society.

Kemerer, C. F., Porter, B. S. (1992). Improving the Reliability of Function Point Measurement: An Empirical Study. Vol 18, No 10 IEEE Transactions on Software Engineering, IEEE Computer Society.

Kitchenham, B., Kansala, K. (1993). Inter-item Correlations among Function Points. Proceedings of the IEEE Metrics Symposium, IEEE Computer Society.

Kitchenham, B., Pfleeger, S.L., Fenton, N. (1994). Towards a Framework for Software Measurement Validation. Vol 21, No 12 IEEE Transactions on Software Engineering, IEEE Computer Society.

Matson, J. E., Barrett, B. E., Mellichamp, J. M (1994). Software Development Cost Estimation Using Function Points. Vol 20, No 4 IEEE Transactions on Software Engineering, IEEE Computer Society.

McCabe, T. J. (1976). A Complexity Measure. Vol 2, No 4 IEEE Transactions on Software Engineering, IEEE Computer Society.

Mukhopadhyay, T., Kekre, S. (1992). Software Effort Models for Early Estimation of Process Control Applications. Vol 18, No 10 IEEE Transactions on Software Engineering, IEEE Computer Society.

Park, R.E., Goethert, W.B., Webb, J.T. (1994). Software Cost and Schedule Estimating: A process Improvement Initiative. Special Report CMU/SEI-94-SR-3, Carnegie Mellon University.

Pressman, R.S. (1992). Software Engineering: A Practitioner's Approach. McGraw-Hill, Inc.

Putnam, L.H., Myers, W. (1992). Measures for Excellence: Reliable Software on Time, within Budget. Englewood Cliffs, NJ: Prentice Hall.

Symons, C.R. (1988). Function Point Analysis: Difficulties and Improvements. Vol 14, No 1 IEEE Transactions on Software Engineering, IEEE Computer Society.

Telecom, (1992). WRS1 Project File Notes. Australian Telecommunications Corporation.

Telecom, (1993). WRS2 Project File Notes. Australian Telecommunications Corporation.

Thomsett, R. (1991). Managing Superlarge Progects: A Contingency Approach. Vol 4, No 6 American Programmer, American Programmer Inc.

Thomsett, R. (1993). Third Wave Project Management. Englewood Cliffs, NJ: Prentice Hall.

Verner, J. Tate G. (1992). A Software Size Model. Vol 18, No 4 IEEE Transactions on Software Engineering, IEEE Computer Society.

Weinberg, G.M. (1971). The Psychology of Computer Programming. New York: Van Nostrand Reinhold.

Weinberg, G.M. (1993). Quality Software Management: Volume 2 First Order Measurement. New York: Dorset House Publishing.

Wydenbach, G., Paynter, J. (1995). Software Project Estimation: A Survey of Practices in New Zealand. Technical report No 97, University of Auckland.

Attachment 1
Modified Wideband Delphi Estimating Sheet

	Dollars			Effort		
	Optimistic	Probable	Pessimistic	Optimistic	Probable	Pessimistic
Problem Definition & Feasibility Study						
Problem Definition						
Study Team						
Consultants						
Feasibility						
Study Team						
Consultants						
Prototyping						
Macro Estimating						
Cost Benefit Analysis						
Reporting						
Total	$ -	$ -	$ -	0	0	0

Requirements Definition

Functional Specification						
Data Specification						
Prototyping						
Infrastructure Specification						
Specification Review						
Documentation						
Customer review						
Tender preparation						
Tender evaluation						
Total	$ -	$ -	$ -	0	0	0

Design

	Dollars			Effort		
	Optimistic	Probable	Pessimistic	Optimistic	Probable	Pessimistic
System specification						
Architectural specification						
Detailed functional specification						
Detailed data specification						
Component design						
Test specification						
System design						
Test design						
Acceptance test spec & design						
Data load spec & design						
Documentation						
Total	$ -	$ -	$ -	0	0	0

Development

Component coding & testing						
Subsystem linking & testing						
System Integration						
Operations and system documentation						
System Interface testing						
System Testing						
User Documentation						
Total	$ -	$ -	$ -	0	0	0

	Dollars			Effort		
	Optimistic	Probable	Pessimistic	Optimistic	Probable	Pessimistic
Implementation						
Prepare & load acceptance tests						
User acceptance test						
Installation						
Data validation						
Database loading						
Manual data loading						
Operations acceptance test						
Fault fixing						
Phase out old system						
Total	$ -	$ -	$ -	0	0	0
Training						
Project Team training						
Training software						
Training hardware						
Design & loading training system						
Preparation of training programme						
Operations training						
User training						
Training environment						
Total	$ -	$ -	$ -	0	0	0

	Dollars			Effort		
	Optimistic	Probable	Pessimistic	Optimistic	Probable	Pessimistic
Development Software & Hardware						
Development Server						
Test Servers						
Development terminals						
Communications						
CASE tools						
Horizontal Software						
Vertical Software						
Total	$ -	$ -	$ -	0	0	0

Operational Software & Hardware						
Servers						
Memory						
Storage						
Processors						
Software						
Terminals						
Memory						
Storage						
Screens						
Processors						
Software						
Communications						
Network capacity						
Network elements						
Software						
Total	$ -	$ -	$ -	0	0	0

		Dollars			Effort	
	Optimistic	Probable	Pessimistic	Optimistic	Probable	Pessimistic
Project Management & Administration						
Project management						
Administration						
Administration hardware & software						
Stationary						
Accommodation						
Travel costs						
Quality (IV & V)						
Configuration management						
Planning & control						
Recruiting costs						
[illegible]	$ -	$ -	$ -	0	0	0
Environmental Changes						
Ergonomic Changes						
Policy Changes						
Procedures						
Development Standards						
Customer Impact						
Total	$ -	$ -	$ -	0	0	0
Grand Total	$ -	$ -	$ -	0	0	0

Project Estimate	**Dollars $ -**	**Manhours 0**