The Applicability of Case-Based Reasoning to Software Cost Estimation

by

Anton Lokotsch

Submitted in partial fulfillment of the requirements for the degree

MSc. (Computer Science)

in the

Faculty of Science and Agriculture
School of Mathematics, Statistics and Information Technology

University of Natal, Pietermarizburg

January 2002

Abstract

The nature and competitiveness of the modern software development industry demands that software engineers be able to make accurate and consistent software cost estimates. Traditionally software cost estimates have been derived with algorithmic cost estimation models such as COCOMO and Function Point Analysis. However, researchers have shown that existing software cost estimation techniques fail to produce accurate and consistent software cost estimates. Improving the reliability of software cost estimates would facilitate cost savings, improved delivery time and better quality software developments. To this end, considerable research has been conducted into finding alternative software cost estimation models that are able produce better quality software cost estimates. Researchers have suggested a number of alternative models to this problem area. One of the most promising alternatives is Case-Based Reasoning (CBR), which is a machine learning paradigm that makes use of past experiences to solve new problems. CBR has been proposed as a solution since it is highly suited to weak theory domains, where the relationships between cause and effect are not well understood.

The aim of this research was to determine the applicability of CBR to software cost estimation. This was accomplished in part through the thorough investigation of the theoretical and practical background to CBR, software cost estimation and current research on CBR applied to software cost estimation. This provided a foundation for the development of experimental CBR software cost estimation models with which an empirical evaluation of this technology applied to software cost estimation was performed. In addition, several regression models were developed, against which the effectiveness of the CBR system could be evaluated. The architecture of the CBR models developed, facilitated the investigation of the effects of case granularity on the quality of the results obtained from them. Traditionally researchers into this field have made use of poorly populated datasets, which did not accurately reflect the true nature of the software development industry. However, for the purposes of this research an extensive database of 300 software development projects was obtained on which these experiments were performed.

The results obtained through experimentation indicated that the CBR models that were developed, performed similarly and in some cases better than those developed by other researchers. In terms of the quality of the results produced, the best CBR model was able to significantly outperform the estimates produced by the best regression model. Also, the effects of increased case granularity was shown to result in better quality predictions made by the CBR models. These promising results experimentally validated CBR as an applicable software cost estimation technique. In addition, it was shown that CBR has a number of methodological advantages over traditional cost estimation techniques.

Preface

The experimental work described in this dissertation was conducted in the School of Mathematics, Statistics and Information Technology, at the University of Natal, Pietermaritzburg, from the 7th of January 2001 to the 21st January 2002 under the supervision of Professor Don Petkov.

These studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any University. Where use has been made of work of others, it has been duly acknowledged in the text.

Acknowledgements

The author wishes to acknowledge the assistance of those mentioned below:

Professor Don Petkov, who supervised and guided this research.

Jean-Marc Desharnais for providing the software development data for this research.

Tecc Inno for providing the CBR Works development environment with which the CBR models were developed.

The School of Mathematics, Statistic and Computer Science of the University of Natal, Pietermaritzburg, for allowing me to make use of their facilities and resources.

Table of Contents

Table of Figures

List of Tables

List of Abbreviations

ACE	Analogy Cost Estimation
ANGEL	ANaloGy SoftwarE tooL
CBR	Case-Based Reasoning
FPA	Function Point Analysis
GSC	General System Characteristic
IFPUG	International Function Point Users Group
KSLOC	Thousands of Source Lines of Code
LOC	Lines of Code
MARE	Mean Absolute Residual Error
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
Pred(25)	Prediction Level for MRE values lower or equal to 25%
SLOC	Source Lines of Code
UFP	Unadjusted Function Point
VAF	Value Adjustment Factor

Chapter 1: Introduction

This research deals with the AI technique of *Case Based Reasoning* applied to the field of *software development effort cost estimation.* This introductory section will provide an overview of these fields.

1.1 Background to the Problem

Accurate project effort prediction is an important goal for the software engineering community (Shepperd and Schofield, 1997). To date most research has focused upon developing algorithmic effort models such as COCOMO (Boehm, 1981) and Function Point Analysis (FPA) (Albrecht, 1979), derived from the statistical analysis of historical project data. These models are often unable to model the complex set of relationships that are evident in many software development environments (Mair et al, 1999). Thus, none of these models have been entirely successful in their estimation accuracy or adoption by the software development industry (Jeffery and Walkerdon, 1999; Schofield, 1998). Software engineers have expressed concern over their inability to accurately estimate costs associated with software development. This concern has become even more pressing as costs associated with software development continue to increase (Kemerer, 1987).

The accuracy of software project estimates has a direct impact on the quality and financial viability of software projects (Mukhopadhyay et al, 1992) and it is essential that the software engineer be able to maximize both of these attributes. For this reason a number of alternative estimation techniques have been proposed. Most of these have been of the Machine Learning genre and include such techniques as Neural Network and Case Based Reasoning (CBR) models. Of these models, CBR has received the most attention by researchers in recent years (Aarnodt and Plaza, 1994).

Software Cost Estimation

There are a number of different techniques for estimating software development effort described in the literature. Two of the most popular models mentioned are COCOMO 81 (Boehm, 1981) and Function Point Analysis (FPA) (Albrecht, 1979). These provide an example of the common distinction between the two large classes of estimation models, namely those that use lines of code (LOC), and those that use functional complexity as a basis to their estimation.

Researchers have suggested many classifications for cost estimation models. Some have chosen to group models into all-encompassing categories such as algorithmic models, which broadly defines models that are derived from historical project data and then use statistical measures to derive an estimate. Irrespective of their classifications, the two most popular models of software cost estimation; COCOMO and FPA have gone through a process of evolutionary improvement, being revised by the authors and other organizations. COCOMO 2000 was developed to take into account the criticisms that have been directed at the original COCOMO 81 model. The focus of this criticism has been the models use of a LOC based metric as its primary estimator (Dolado, 1997). Similarly the original FPA model has been regularly revised and in its present form is regulated by the International Function Point Users Group (IFPUG).

Machine Learning techniques as applied to software cost estimation models are a recent development in this problem domain. Two of the most prominent techniques applied are Neural Networks and Case-Based Reasoning (Mair et al, 1999). Both have shown considerable promise as alternatives to the traditional methods of effort estimation.

An Overview of Case-Based Reasoning

Case-Based Reasoning (CBR) is a problem-solving paradigm based on the reuse of past experiences. There exists considerable optimism about the use of CBR in difficult

problem solving areas where human expertise is evidently experience based (Delany, 2000). CBR is said to be particularly well suited to weak theory domains, where the relation between cause and effect is not well defined or understood. In this respect it is fundamentally different to other AI approaches in two ways:

- Aarnodt and Plaza (1994) state that CBR does not rely solely on the general knowledge of a problem domain or make associations along generalized relationships between problem descriptors and conclusions. Instead, CBR is able to utilize the specific knowledge of previously experienced problem situations. Each instance of a previously solved problem is viewed as knowledge container or case, which a CBR system is able to utilize in solving new problem situations.
- Secondly, CBR is an approach to incremental sustained learning, since a new experience is retained each time a problem is solved.

CBR solves new problems by finding a similar past case and reusing it in the context of the new problem. This process is described in the CBR cycle as suggested by Aarnodt and Plaza (1994):

- Retrieve the most similar case or cases.
- Reuse the information and knowledge contained in that case to solve the new problem.
- Revise the proposed solution.
- Retain the solution or parts thereof to facilitate future problem solving.

The use of CBR was first suggested by Schank (1977). His work culminated in the development of the first CBR system by Kolodner (1983). Since then, CBR has been successfully applied to such varied fields as diagnosis, arbitration, legal reasoning, design and repair.

A number of CBR tools have been developed to aid the development of CBR applications. This allows the developer to concentrate on the specifics of the application rather than having to reformulate the basic principles of CBR.

More recently CBR has been suggested for use as a software effort estimation tool. This will be discussed in more detail in the following section on CBR applied to software effort estimation.

Case Based Reasoning Applied to the Field of Software Cost Estimation

Existing estimation techniques have failed to produce consistently accurate software development effort estimates. Mukhopadhyay et al (1992) argue that these approaches are not incorrect, but rather insufficient and that further insight into the problem is required. The developers of these models have attempted to derive models that quantify the causal dependencies within the domain. Since these models do not consistently solve the problem accurately, it can be assumed that they do not model the problem domain effectively since the causal relation between effort and input variables is not well understood (Delany, 2000). However, the use of CBR avoids the need to model the problem domain. Furthermore CBR is able to produce estimates in situations where little domain knowledge exists and is able to augment its knowledge base with solved solutions that may be relevant to future problems.

Mukhopadhyay et al (1992) were the first to apply CBR to software effort estimation. Since then a number of similar systems have been successfully applied to this field. Mukhopadhyay et al (1992), Shepperd and Schofield (1995), Finnie and Wittig (1997) have all developed CBR systems which produce estimates with accuracy that exceed those of traditional estimation techniques. Previous research is thus used as a basis to the research and experimentation conducted in this thesis.

1.2 Goals and Sub-Goals of the Research

The primary goal of this research is to investigate the applicability of CBR to the field of software cost estimation.

This primary goal can be decomposed into a number of sub-goals. These sub-goals can objectively be divided into theoretical and experimental components.

The theoretical sub-goals can be formulated as follows:

- To provide insight into CBR through describing the technology, methodologies and applications.
- To formulate a categorization of software cost estimation techniques.
- To provide an overview of software cost estimation techniques and their methodologies.
- To provide an overview of CBR applied to software cost estimation.
- To define the functional requirements of a CBR tool.
- To review the CBR tools considered for this research.
- To conduct experimental validation of the application of CBR to software cost estimation. The experimental sub-goals can be formulated as follows:
    - To develop a CBR system suitable for software cost estimation.
    - To define an appropriate case retrieval mechanism.
    - To suggest case adaptation mechanisms for the experimental CBR system.
    - To suggest and define appropriate measures of accuracy for the CBR system.
    - To determine the effect of case granularity on the quality of estimations made by the experimental CBR system.

- To report on the success and applicability of the CBR system as a solution for software cost estimation.

The above goals can be justified on the basis of the current state of research in the areas of software cost estimation and CBR.

1.3 Scope and Delimitations of the Research

The research conducted establishes the theoretical foundations of CBR and software cost estimation and discusses the application of CBR to the latter. This provides momentum for the development of an experiment which would produce results from which conclusions can be drawn as to the applicability of CBR to software cost estimation.

The discussion on software cost estimation will focus on two of the most prominent and applied estimation methods used, namely:

- COCOMO 81 (Boehm, 1981) and COCOMO 2000 (Boehm et al, 2000)
- Function Point Analysis (FPA) (Albrecht, 1979) and IFPUG FPA version 4 (IFPUG, 1996).

The discussion of Function Point Analysis has a direct impact on the experimental component of this research, since it is the primary method used in compiling historical project data. As mentioned in the background discussion into CBR, the use of past project data forms the basis of an estimation made by a CBR system. This influences the design of the experiment, since the formulation of this data needs to be analyzed so as to determine appropriate case structures for the experimental CBR system.

Many studies in the field of cost estimation are limited in scope as their conclusions are based on the results obtained from small and medium sized datasets (Wieczorek, 2001), which often do not reflecting cross-organizational influences. This research aims to overcome this limitation by making use of a more comprehensive dataset that is more reflective of the software development industry. At the same time, a delimitation of this

research is that it is accepted that the results obtained though experimentation would be highly dependant on the input data provided to the CBR system developed in the experiment. The quality of this data cannot be verified since project data is collected by different software engineers in different organizations. Thus, the author of the dataset used in this research has no control over the robustness of the compiled data. Although the collection methods used are standardized, this cannot completely assure data consistency, since software development projects are of a highly specialized nature that does not completely allow for generalizations across organizations or application areas.

The applicability of CBR to software cost estimation will be determined on the basis of the discussion on CBR applied to software cost estimation, the results obtained through experimentation and comparisons made to other research in this field. Comparison is however limited to the small pool of published research available in this field.

1.4 Research Methodology

According to Robey (1996), justification of research can be made by suitably linking the research aim to the theoretical foundation of the research and research methods used. In addition to this, the theoretical foundation of the research determines the research methods used. Robey (1996) refers to this as the triad for justification of research. The relationships between the three components are illustrated in Figure 1.0 (Adapted from Robey, 1996).

The primary research aim is to investigate the applicability of CBR to software cost estimation. This goal determines the methodologies used and this necessitated an investigation into CBR, software cost estimation techniques and CBR applied to the field of software cost estimation. This was conducted in the form of literature reviews on the above topics. This establishes the theoretical foundation upon which the experimental aspect of this research is conducted, as described below.

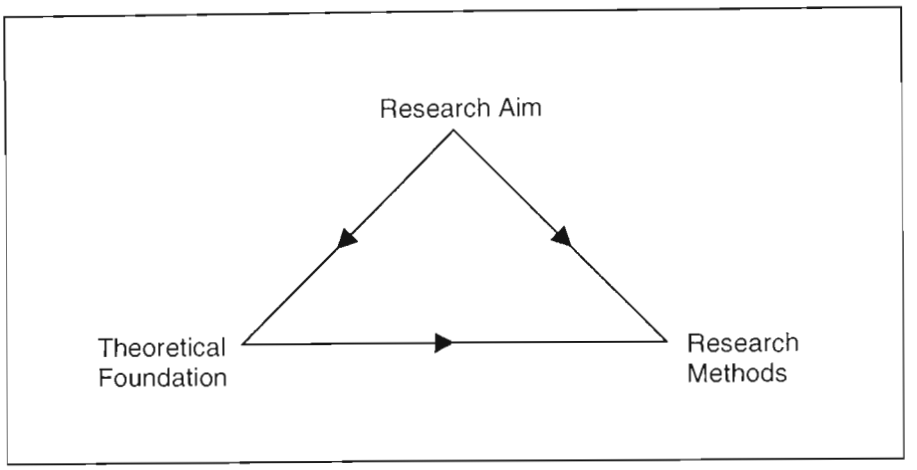

Figure 1.0 Triad for the justification of research (Robey, 1996)

The applicability of CBR applied to the field of software cost estimation can at best be determined using the experimental methodology. This is because, to the best knowledge of the author, internationally there are no companies that are applying CBR to software cost estimation. Therefore no field investigation or action research is possible in this area. Since several datasets were reported to have been used by other researchers in this field, it was hoped that at least one of them would be made available for the experiment.

This research aimed to produce replicable results by detailing the methods used in formulating the CBR system used for experimentation. To adhere to this, experimentation in this research was conducted through the application of a consistent procedure of development, structuring and analysis of results. As in most experimental fields, replication is the key to establishing validity and generalizability of results. Criticism has been directed at researchers for failing to detail the methods used and formulate the results in a manner that allows for comparison to other research (Wieczonek, 2001). For example, Finnie and Wittig (1997) produce results and an experimental methodology that allow for replication and comparison for future researchers. In addition to the above discussion, experimental verification and validation of the experiment was enhanced through the application of statistical analysis of the results.

1.5 Importance of This Research to the Fields of Case-Based Reasoning and Software Cost Estimation

The need for accurate software cost estimates has been highlighted in the introductory section of this chapter. CBR is a new approach to problem solving and learning (Aarnodt and Plaza, 1994) that has received much attention from researchers. Furthermore, CBR as applied to the field of software cost estimation is comparatively an even newer field of study in comparison to other cost estimation techniques such as Function Point Analysis (since 1979) and COCOMO (since 1981). As mentioned in the section on project scope, the studies on the use of CBR applied to software cost estimation that have been completed are often inconclusive (Wieczonek, 2001), owing to the nature of the data used in the experiments conducted. This indicates a need to perform further research into the applicability to this field using data that is more representative of real world software developments. To the best knowledge of the author, no previous research has been conducted on the effects that granularity of data has on the accuracy of a CBR systems predictions.

An added justification of this research in the context of software cost estimation is that an attempt is made to derive a categorization of the various software cost estimation techniques, placing them in context with modern methods of estimation. This is a facet of software cost estimation which is often avoided in the literature.

This research therefore intends to make both a theoretical and practical contribution to the fields of software cost estimation and case based reasoning.

1.6 Overview of the Structure of the Dissertation

Since the purpose of this research is to determine the applicability of CBR to software cost estimation, a number of theoretical chapters precede the experimental chapter to establish the appropriate theoretical foundations.

Chapter 2 provides an overview of the AI paradigm of CBR by discussing the involved technologies, applications and methodologies. This is followed by an overview of the various software cost estimation techniques in Chapter 3. This chapter intends to define a categorization of the various cost estimation models and uses this as a framework for the rest of the chapter.

The use of CBR in the field of software cost estimation is discussed in Chapter 4. This chapter will include information on existing CBR systems developed for this purpose. A discussion is also undertaken on the functional requirements that a generic CBR tool should have. This culminates in an examination of the CBR tools short listed for the development of an experimental CBR system.

Chapter 5 focuses on the research component of this research; the methodologies used, experimental design and nature of the experiments are discussed in detail. Included in this chapter is a discussion of the experimental results, which intends to place this research in the same context as similar research and seeks to compare and evaluate the results obtained.

Chapter 6 is the concluding chapter in which the theoretical and practical contributions of this research to the software engineering community will be discussed. Suggestions will be made as to possible future directions of research into CBR applied to software cost estimation.

Chapter 2: Case-Based Reasoning

2.1 The Origins of Case-Based Reasoning

Case-Based Reasoning (CBR) is a problem-solving paradigm based on the reuse of past experiences that has developed into a well-established sub-field of Artificial Intelligence (Leake, 1996). CBR is classified as a machine learning paradigm since it embodies some of the facets of the human mind that allows for the rapid solution of hugely complex problems (Schank, 1982). Many successful implementations of CBR are in existence in commercial fields worldwide, solving problems ranging from the operational to the organizational levels (Kolodner, 1993). CBR has been in existence for approximately two decades and now enjoys the level of refinement that makes it superior to other knowledge based reasoning techniques (Watson and Marir, 1994). The advantages of CBR over traditional knowledge based techniques are discussed in Section 3.3.

It is widely accepted that CBR has its philosophical origins in the field of cognitive science, since the basic concepts and the operation of a CBR system attempt to simulate an experienced and intelligent human type interaction (and response) to a given problem. The study of CBR is driven by two primary motivations: The first from cognitive science is the desire to model human reasoning and learning. The second, from artificial intelligence, is the pragmatic desire to develop the technology to make AI systems more effective (Leake, 1996). The conceptual mechanisms used in CBR can be traced to the fields of psychology and philosophy, which have investigated and theorised on concept formation, experiential learning and problem solving (Wittgenstein, 1953; Tulving, 1972; Smith et al, 1981). Wittgenstein (1953) observed that natural physical concepts are in fact polymorphic and therefore cannot be classified by a single set of necessary features. They can instead be defined by a set of instances (cases) with family resemblances (similarities). Aarnodt and Plaza (1994) site this work as the philosophical basis for CBR.

It is widely accepted that the first published semblance of CBR came from Schank and Abelson (1977), who suggested the use of scripts for knowledge representation. Their paper "Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures" was foundational for the CBR field. Schank persevered with the exploration of the role that memory of previous situations and situation patterns have in problem solving and learning. This research culminated in the publication of his paper on "Dynamic Memory: A Theory of Learning in Computers and People" (Schank, 1982). This was the first work that described a memory-based approach to reasoning and suggested an architecture for developing such a reasoning system. This would be the basis for CBR memory simulation and implementation.

A major advancement in the field of CBR occurred in 1983 with the development of the first computer implemented CBR system, CYRUS (Computerised Yale Retrieval and Update System) (Kolodner, 1983). CYRUS implemented Schanks dynamic memory model using Memory Organisation Packets (MOP's) for problem solving and learning. CYRUS was essentially a question answering system which retained knowledge of US Senator and former secretary of state, Cyrus Vance's travels and agenda.

In the subsequent decade following Schanks' foundational work and Kolodner's CBR implementation, came a proliferation of academic experimental CBR systems. The dynamic memory model used in CYRUS has served as the basis for many of these CBR systems, most notably: CHEF (Hammond, 1986), PERSUADER (Sycara, 1987), MEDIATOR (Simpson, 1985), CASEY (Koton, 1992) and JULIA (Hinrichs, 1992).

Schank's dynamic memory model was however not the only approach to case storage and management in CBR systems. Porter and Bareiss (1986) suggested an alternative approach to Schank's dynamic memory model. This became known as the category-exemplar model. Their CBR system known as PROTOS, unified general domain knowledge and specific case knowledge into a single case memory model (Watson and Marir, 1994). This work was later furthered and implemented by Banting (1991) in a

CBR system known as GREBE. The dynamic memory model and the category-exemplar model are discussed further in Section 3.1.

The academic CBR systems referred to above, increased markedly in sophistication as CBR became accepted as an effective solution to knowledge based problems. The maturity that CBR was attaining culminated in the first commercial application of CBR: CLAVIER (Barletta and Henessy, 1989). CLAVIER was developed at Lockheed to assist in the manufacturing process of aircraft materials. CLAVIER has been continually improved over the course of its operation since its initial implementation and now makes use of a case base of over 300 entries – far more than the initial 20 cases. Other prominent successful CBR implementations are: CaseLine, used by British airways for fault diagnosis on certain model aircraft and SMART (Acorn and Walden, 1992) as used by Compaq as a help-desk application to solve common computer problems.

Many other successful commercial applications of CBR exist. These are indicative of the success that CBR has achieved in diverse problem areas such as: diagnosis, decision support, design, planning and legal reasoning. The areas of application of CBR have become a form of classification for CBR systems. This implies that the implementations of CBR in working systems can be organised according to the particular field of application. A listing and description of a number of prominent CBR implementations is outlined in Table 2.1. A discussion of the various implementations CBR falls beyond the scope of this thesis as there are over 100 commercially implemented CBR systems (Aarnodt and Plaza, 1994) and countless academic demonstrators (Kolodner, 1993).

Owing to the diversity of the possible applications of CBR it is evident that CBR's flexibility has in the past been a hindrance to its widespread adoption. Each CBR application would have to be specifically tailored to its domain of application and then further to its actual implementation. This required a customised CBR system for each application. The maturity that the field of CBR has attained and the demand for a platform for CBR system development, resulted in the development of a number of CBR shells. ESTEEM, CBR Works (Tecc Inno) and REMIND (Inference Corp) are a few of

the CBR shells that have been commercially developed. CBR shells have allowed for the easier development and deployment of CBR systems in both the academic and commercial environments. These CBR shells are discussed further in Chapter 4.

Table 2.1 Table of selected CBR systems classified according to area of application (Kolodner, 1993; Watson and Marir, 1994).

Field	Name	Date	Brief Description
Knowledge acquisition	**REFINER**	1988	Helps experts refine their knowledge
Legal reasoning	**JUDGE**	1989	Determines criminal sentences
	HYPO	1991	Creates legal arguments
	GREBE	1991	Classifies new cases
	KICS	1994	Interprets building regulations
Explanation of anomalies	**SWALE**	1986	Gives explanations for the deaths of humans and animals.
Diagnosis	**PROTOS**	1986	Diagnoses hearing disorders
	CASEY	1989	Diagnoses potential heart failure
	CASCADE	1992	Diagnoses operating system crashes
	PAKAR	1994	Diagnoses possible building defects
	SMART	1993	Help desk assistant for diagnosing common PC problems
Arbitration	**MEDIATOR**	1985	Suggests resolutions to disputes
	PERSUADER	1987	Mediator for union-management disputes
Design	**CYCLOPS**	1989	Solve landscaping design problems
	JULIA	1992	Designs and plans meals
	CADET	1992	Mechanical design assistant
	ARCHIE	1992	Architectural design assistant
	CLAVIER	1989	Autoclave layout assistant
Planning	**BATTLE**	1989	Land warfare strategist
	BOLERO	1993	Diagnostic disease planner
	TOTLEC	1993	Solves manufacturing planning problems
	JULIA	1989	Meal planner
Repair and adaptation	**CHEF**	1986	Creates new recipes
	PLEXUS	1986	Adapts existing routines
	COACH	1987	Football play strategist
Tutoring	**DECIDER**	1987	Gives guidance according to goals
Software Cost Estimation	**ESTOR**	1992	Estimation of software cost from historical data on software projects.
	MERMAID	Unknown	Software cost estimation tool
	ACE	1999	Research cost estimation model
	ANGEL	1995	An automated environment for developing CBR cost estimation applications.

2.2 The Process of Case-Based Reasoning

Aarnodt and Plaza (1994) prescribe a two-part framework for describing CBR methods and systems. This involves the description of the CBR cycle and a task-method structure for CBR. The first is a model that identifies the main sub-processes of the CBR cycle and their interdependencies, while the second is a task-oriented view where the problem solving methods are described. This structure is followed in the discussion on the CBR process below.

2.2.1 The CBR Cycle

A case-based reasoner solves new problems by adapting solutions that were used successfully in the past to solve old problems (Riesbeck and Schank, 1989). From a human perspective the process of reasoning through past experience does not require any thought or formal method. However, it is necessary to have some structure to the reasoning process to formalise the way in which CBR can be applied and used in the more artificial computing environment. This process is central to all CBR systems and can be described as a CBR cycle.

Similarly to Kolodner (1993), Aarnodt and Plaza (1994) propose that the process of CBR is a four phase cyclical process:

- RETRIEVE the most similar case(s)
- REUSE the case(s) to attempt to solve the problem
- REVISE the proposed solution as necessary
- RETAIN the new solution or case

When a problem is encountered, previously experienced cases are retrieved from memory. The retrieved case(s) are then reused and possibly revised to solve the problem. The new solved experience is then retained in the case base. The CBR cycle is depicted in Figure 2.1.

This cyclical view of CBR is a process model designed to give an overview of the CBR process. An alternative, introspective view of the CBR process can be made using a task-method structure. This is appropriate for discussing the operation of a CBR system in more detail since each task is decomposed into its component methods.

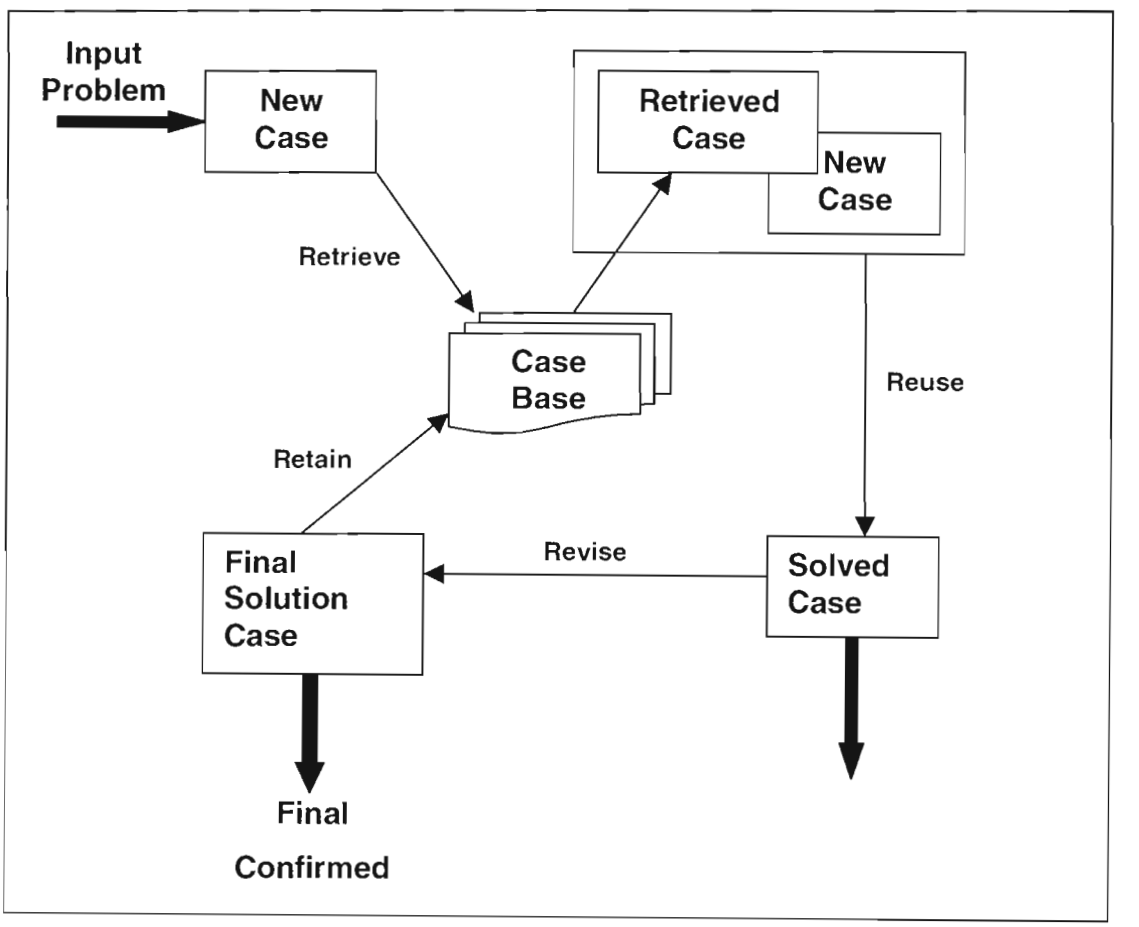

Figure 2.1 Cyclical process of CBR (adapted from Aarnodt and Plaza, 1994)

2.2.2 Task Structure of a CBR System

The task-method decomposition structure of CBR is described graphically in Figure 2.2 (adapted from Aarnodt and Plaza, 1994). There are four primary tasks in the CBR system as mentioned in the CBR cycle above. They are: retrieve, revise, reuse and retain. Each

shall be described in detail to provide a more detailed operational knowledge of a CBR system.

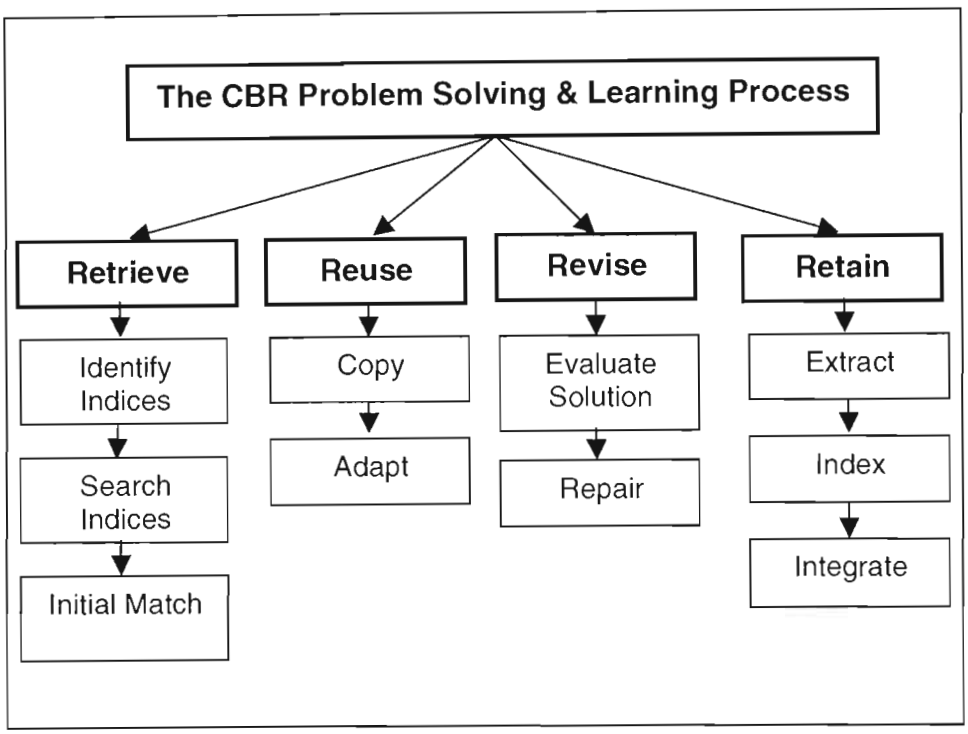

Figure 2.2 Task decomposition of CBR (adapted from Aarnodt and Plaza, 1994)

2.2.2.1 Case Retrieval

The retrieval of a case is vital to the success of the remaining CBR tasks. It is here that an often vague/partially formulated problem description has to be transformed into a comparable case, i.e. the problem case is required to be indexed to make it compatible with the case base. A search procedure then follows, eventually retrieving a case matching or a similarity matching to those cases in the case base.

A number of methods/sub-tasks can be identified as constituting the core activities of the retrieval task. The task methods are:

- Identify Indices

- Search Indices
- Find an Initial match

Identifying appropriate indices

Taking an abstract problem and organising it into a useable format for comparison (and ultimately problem solving) is the primary objective of this method (Aarnodt and Plaza, 1994). In order to accomplish this, the information in the given problem needs to be categorised (using the appropriate CBR method for this system). This means that the problem case is re-constructed into a compatible format using certain criteria to define its structure. The two divergent methods of indexing a problem either use the problems syntactic properties or the semantic properties as primary identifiers. In either case the new problem will be formalised as best possible to facilitate searching and matching with the new cases problem descriptors (indices) serving as input for the searching and matching tasks.

Searching and matching

How the case base is searched depends on the case memory structure (Watson and Marir, 1994). Case memory structure is discussed in detail in Section 3.1. Usually dominant features of the search case would be used to refine the search early on to reduce overhead. This becomes more important for the operation of the CBR system as the size of the case base increases (usually in commercial circumstances). It is here that algorithms are employed to perform the search and matching. The choice of algorithm can be influenced by many factors: size of case base, the nature of indices and overhead versus accuracy requirements of the CBR system. Through the amalgamation of information from the literature a typical set of algorithms used for searching and matching are (Watson and Marir, 1994; Shepperd and Schofield, 1997: Kolodner, 1993):

- Induction
- Nearest Neighbour

- Template Retrieval
- Knowledge Guided Induction
- Fuzzy similarity
- Recency Preference

Nearest neighbour algorithms are by far the most popular form of finding matching cases (Shepperd and Schofield, 1997). Due to the popularity of this type of matching algorithm, this particular approach will be discussed in greater detail than the other methods.

Induction

Induction algorithms develop a decision tree structure from the most pertinent features that discriminate cases. This algorithm is most applicable to CBR systems that require a single case feature as a solution, and where that case feature is dependent upon others (Watson and Marir, 1994, pg10).

Nearest Neighbour

The Nearest Neighbour algorithm is one of the most popular algorithms presented here: Compaq SMART System (Acorn and Walden, 1992) and BROADWAY (Skalk, 1992) are just a few CBR systems that use this algorithm. A prerequisite for the use of this algorithm is the assignment of weights to case features. Nearest neighbour algorithms can be implemented in two ways: as a straightforward distance measurement or as the sum of the squares of the differences for each variable. Either way, this algorithm makes a similarity assessment of the weighted sum of features of the input case to the cases in the case base: each variable must first be standardised so that it has an equal influence in the algorithm; each variable must be weighted according to the degree of influence attached to that feature.

A typical algorithm suggested by Aha (1991) is shown in equations 2.1 and 2.2:

$$SIM(C_1, C_2, P) = \frac{1}{\sqrt{\sum_{i \in P} Feature\_dissimilarity(C_{1i}, C_{2i})}} \quad (2.1)$$

Where, P is the set of n features, C_1 and C_2 are cases and i is the particular case feature. *Feature_dissimilarity* is calculated as follows:

$$Feature\_dissimilarity(C_{1i}, C_{2i}) \begin{cases} (C_{1i}, C_{2i})^2, \text{if the features are numeric} \\ 0, \text{if the features are categorical and } C_{1i} = C_{2i} \text{ or 3} \\ 1, \text{where the features are categorical and } C_{1i} \neq C_{2i} \end{cases} \quad (2.2)$$

A straightforward distance measurement for determining feature similarity as suggested by Petrak et al (1994) is presented in Equation 2.3 below.

$$similarity(C_{1i}, C_{2i}) = 1 - \frac{|C_{1i} - C_{2i}|}{C_{1i}\ \max - C_{2i}\ \max} \quad (2.3)$$

Where, as in Equation 2.1 and 2.2, C_1 and C_2 are cases and i is the particular case feature. C_{1i} *max* and C_{2i} *max* are the maximum possible values for those features.

The form of the Nearest Neighbour algorithm may be implementation specific but the fundamental operation will always remain the same. The overhead involved in using the Nearest Neighbour algorithm is significant. Retrieval time increases linearly with the number of cases. It is therefore preferable to use this algorithm on a relatively small case base if there are performance requirements that need to be fulfilled. To overcome the performance limitations of this algorithm, it is often used in conjunction with another less precise algorithm such as Template Retrieval mentioned below. This allows the search base to be narrowed before the costly Nearest Neighbour algorithm is then used to refine the search.

Template Retrieval

Template retrieval algorithms use a feature comparison method to retrieve and match cases. Here all cases that fit the input feature criteria are returned. This can be likened to a typical database query (Watson and Marir, 1994). Template retrieval is often used to narrow the search base for later more detailed search procedures. Most often Template Retrieval is used as a precursor to nearest neighbour retrieval. In Template Retrieval the user can supply ranges and all cases that match are retrieved (Shepperd and Schofield, 1997).

Knowledge Guided Induction

Knowledge guided induction introduces suggestive knowledge in manually identifying case features that are considered important or influential on the primary case feature. This algorithm is often used in combination with other algorithms and is particularly suited for large case bases, since an informed decision can vastly reduce the search base (Watson and Marir, 1994).

Fuzzy Similarity

In Fuzzy similarity, fuzzy concepts such as at-least-as-similar and just-noticeable-difference are employed. (Shepperd and Schofield, 1997). These rules are formed and the retrieval process then tries to meet their requirements in finding the closest matching features.

Recency Preference

In Recency preference algorithms, preference is given to recently matched cases over those that haven't been matched for a defined period of time or frequency (Shepperd and Schofield, 1997).

The abovementioned similarity measures suffer from a number of disadvantages (Shepperd and Schofield, 1997):

- The major disadvantage being that they are computationally intensive. As such a number of more efficient algorithms have been proposed by researchers such as Aha (1991).
- The algorithms are mostly intolerant of noise and irrelevant features. This can however be controlled through the stricter implementation of weights.
- Symbolic and categorical features are problematic for similarity algorithms. Although several algorithms have been proposed to overcome this, none have met with any particular success.
- These similarity measures fail to recognise higher order similarities in the structure of the data.

2.2.2.2 Case Reuse

As mentioned previously, the primary objective of CBR is to use past cases to solve new problems. The Case Reuse task aims to do this by reusing the knowledge contained in the retrieved case(s) in the context of the new case. In order to accomplish this it is necessary to note the differences between the retrieved case and the new case and then establish which portion of the retrieved case can be applied to the new case. This is popularly referred to as case adaptation. Shepperd and Schofield (1997) state that adaptation is viewed as the most challenging task in CBR research.

If the cases are very similar, then Case Reuse is a fairly simple process of copying the useable solutions of the retrieved case to the new case. This is however not always the situation and changes might need to be made to the solutions provided by the retrieved case to make them apply to the new case.

According to Aarnodt and Plaza (1994), case reuse can be divided into two sub-tasks, namely copy the retrieved case and adapt the retrieved case if necessary:

Copy Retrieved Case

Copy represents the most basic form of Case Reuse. Here, the differences in the cases are not considered a priority, and the solution class of the retrieved case is transferred to the new case as its solution class. In this situation the similarities between the cases is considered more important than the differences between the cases. However, it is often necessary to take these differences into account if a practical solution to the new problem is to be found. The Case Adaptation sub-task is responsible for analyzing the differences and taking them into account when suggesting an informed solution.

Adapt Retrieved Case

The purpose of Case Adaptation is to make the solution contained in the retrieved case(s) applicable to the current problem case. This is done by noting the differences between the problem and solution cases and then applying rules or algorithms to get a resultant solution that has factored in the inherent differences. This ensures that the suggested solution is placed in the same context as the problem.

There are two different techniques that can be used for case adaptation (Aarnodt and Plaza, 1994):

- Transformational Reuse (otherwise known as Structural Adaptation)
- Derivational reuse

Transformational Reuse uses a transform operator T to convert the existing solution to an applicable solution. This can be likened to the operation of transformational matrices. In more concise terms, Transformational Reuse concentrates on the relative equivalence of solutions.

Derivational Reuse applies the successful operators (algorithms, methods or rules) used in arriving at a successful solution in the retrieved case to arriving at a suggested solution for the new case. Thus any unsuccessful operators would be avoided when applying the operators to the new case.

Many varying implementations of the above two methods exist in the literature. Watson and Marir (1994) and Koldoner (1993) categorise the various techniques slightly differently. This indicates a lack of consensus on a formal classification. An adapted classification based on adaptation mechanisms that exist in CBR systems and in the literature is:

Null adaptation

This is the simplest form of adaptation and involves simply copying the retrieved solution to the new case, without making any form of adaptation. This is particularly suited to complex problems that have simple solutions.

Parameter Adjustment

Parameter adjustment is a structural adaptation technique, which uses parameter comparison to decide on what adjustments to the retrieved solution should be made. PERSUADER (Sycara, 1987), creates initial solutions, critically evaluates them, and then suggests more complicated solutions. Similarly JUDGE (Bain, 1986) recommends a shorter sentence for a criminal where the crime is less violent.

Local Search (also referred to as Abstraction and Respecialization)

This is a structural adaptation technique as used in PLEXUS (Alterman, 1986). PLEXUS is a planning system that makes use of this technique. It has an execution time adaptation technique that attempts to find substitute actions for those that cannot be performed. Local Search implements a process of searching in an abstraction hierarchy in the environs of a concept for some close relative that could be substituted for it. It is

generally used when the retrieved case is almost the same as the problem case and could be made to fit with a minor substitution. For example in the case of PLEXUS, a subway ticket station could simply be replaced by a ticket vending machine.

Reinstantiation

Reinstantiation is used to instantiate features of an old solution with new features (Watson and Marir, 1994). MEDIATOR (Simpson, 1985) uses the Reinstantiation method for case adaptation. Reinstantiation can be used when the framework of the new problem is the same as that of the suggested solution, but where the semantics may differ. This implies the possibility for the transfer of direct knowledge from an existing case to the retrieved case if that knowledge is in the same context. CHEF (Hammond, 1986) also uses a Reinstantiation adaptation mechanism. Here a reinstantiation can occur in a recipe where certain ingredients are not available. In the case where a similar recipe exists, the transfer of alternatives to the missing ingredients can take place.

Case-Based Substitution

Case-Based Substitution attempts to find a substitute case by finding a case that can suggest an alternative. CLAVIER (Barletta, 1989) uses case-based substitution.

Problems with adaptation, in particular varying success rates has led to many CBR systems not using the technical adaptation techniques listed above. Instead, many depend on human interaction to adapt and evaluate solutions. This is particularly the case with CLAVIER, which in its latest version allows the user to perform manual adaptation (Kolodner, 1993). Kolodner (1993) states that case adaptation is an area of CBR that is most often ignored by CBR systems. Instead, this is usually left to an expert to make the final adjustments to the retrieved case. The abovementioned adaptation techniques are not representative of the population of adaptation techniques that exist and serve only as an indication of the variety of techniques that have been used in CBR systems.

2.2.2.3 Case Revision

CBR not only reuses past solutions but is also able to learn from mismatches or mistakes if necessary. This is the essence of the Case Revision task and involves two sub tasks, namely; Evaluate and Repair. In these tasks, the suggested solution generated by the Case Reuse task is evaluated for possible learning or is repaired after fault diagnosis.

Evaluate the Proposed Solution

To evaluate the proposed solution requires the testing of the solution in its field of application, i.e. during the course of a CBR systems operation. Owing to the nature of CBR applications, it can take considerable time for the analysis of the solution and thus for the relative success/failure of the solution to become apparent. This is one of the dangers of a CBR system, since the success of the evaluated solution can seldom be guaranteed. Some CBR models overcome this by using an internal model simulation program to evaluate a solution (one such CBR system is CHEF (Bain, 1986)). Once a sufficient evaluation outcome is available, the CBR system then determines if the solution was appropriate or whether it needs to be repaired.

Repair the Proposed Solution

In order to repair a solution, it is necessary to have acquired knowledge on the errors that the solution produced and their exact nature (i.e. being able to produce explanations for them). Explanations can inter-alia can be in the form of reasons why the goals of the solution were not achieved and may include what solution steps in the suggested case were responsible for this. Once the failures and their explanations have been established, the suggested solution can be repaired so as to avoid the undesired outcomes. This can be done manually or as in the case of CHEF (Bain, 1986), a repair module can be used. The sophisticated repair module in CHEF possesses general causal knowledge and domain knowledge on how to avoid or compensate for the undesired outcomes (Aarnodt and Plaza, 1994; Kolodner, 1993). This module may affect changes to the solution that

include additional compensating steps. CHEF is able to learn problem causing circumstances and the appropriate solutions to that problem. After the case is repaired it can either be directly retained in the case memory or re-considered by submitting it to the evaluation-repair procedure.

2.2.2.4 Case Retention

Case Retention is the 'learning' module of CBR. The ability for a CBR system to learn from past experiences and mistakes makes CBR more effective than other traditional knowledge based systems. Retention is dependant on the results of the Case Revision task. The Retention task is responsible for selecting what information from the new solution to retain (if not all of it), in what format the new information is to be retained and how this is integrated in the memory storage schema. It follows that there are three sub-tasks involved here, namely; Extract, Index, and Integrate.

Extract Information from the Solution

Learning is an integral part of many CBR systems and improves the performance or effectiveness of the system. Deciding what information to Extract from the new case solution will have an effect on the problem solving abilities of the CBR system in future cases. Thus when a new case is solved it is as often beneficial that some form of 'learning' through retention occurs.

The form of retention will vary according to how the problem was solved or under what circumstances the solution was created. If a new case was developed using knowledge from a previous case, then in order to retain the new knowledge, either a new case can be created in the case memory or a more generalized form of the existing case can be created (this is especially considered if the two cases are highly similar in context and outcome). If the new case was solved through outside intervention (for example, from a user or system operator), then it is most likely that a unique case would be created in memory. Often the information regarding the solution finding process obtained/recorded in the

Case Revision task is retained to avoid future similar failures or help solve them. This information may be stored as case-specific information (tagged onto the case) or stored separately in the case base. Information regarding the problem solving technique used can also be stored in the case base so as to expedite the solving of future similar cases. This can include information on what problem solving technique was used for that specific case. In addition explanations can be included as to why a specific solution was used.

It is evident from the discussion above that the information stored in the case base can go beyond problem descriptors and solutions. The form and type of data stored is dependant on the area of application of the CBR system and on the requirements set by the users. For example: a "strategic military decision support" CBR system (a hypothetical CBR system) would require that each solution have associated explanations with detailed probable failures associated with each case. This can be contrasted with a hypothetical CBR "automobile diagnosis system", which would have limited need for detailed explanations, since the solutions would most likely be considered exact.

Index the Case to be Retained

The task of indexing involves deciding on what type of indices should be used for the cases and how the search space of the indices should be structured. This has a defining effect on how cases are retrieved, stored and manipulated in a CBR system. As uncomplicated as this seems, the problem that this poses is referred to as the Indexing problem. Kolodner (1993) refers to the retrieval of appropriate cases as the single biggest issue in CBR.

Indexing is a multi-part process: the first being the identification and subsequent labelling of cases on entry into the case base; the other is that of organizing the indexed cases in a manner that would allow for efficient searches through the case base. The choice of indexing system for the case base has repercussions on the performance and effectiveness (the ability to produce useful solutions) of a CBR system.

The indexing task usually requires that the most pertinent features of a case be identified and then labels assigned to them. This ensures that only the dominant features of a case are initially searchable and allows for a structured form of memory usage. There are a number of memory storage strategies that can be used. These will be discussed in Case Memory Representation. In the most trivial case of indexing, all input features of a case can be indexed. Matches would have to be performed on every single feature of every case. This can be effective in simplistic problem solving circumstances, but can also result in overburdening the case memory, especially as the size of the case base increases. Once a decision has been made on what features of a case to index, the case base can be updated with the new information in the Integrate task.

Integrate the New Case Into the Case Base

Knowledge integration is the last remaining sub-task of the Retention task and indeed of the CBR process. The knowledge gained from a new case is stored in the case memory in whatever form has been decided by the Extract and Indexing sub-tasks.

Integration of case data may involve the modification or re-organisation of the existing case storage structure. This can occur if a continual improvement policy is implemented by the Retention task. This continual improvement policy is an important component of the learning mechanism of CBR and involves improving the application of indexing method used. This adjustment is determined by the relative success or failure of the cases used to solve a problem. For example, if it is found that certain indexed features of a case are more relevant for solving problems, then the importance of those features can be emphasised when being indexed. The opposite can also occur where features that are found to be less critical in providing a successful solution outcome are given a weaker association to their parent case. This will be discussed further in Case Memory Representation.

2.3 Case Memory Representation

Kolodner (1993) identified case memory as one of the vital components of a CBR system, since the case memory structure determines how the cases will be stored for retrieval and retention purposes. Two prominent case memory representation techniques will be explained in more detail in this section.

2.3.1 Case Memory Representation Techniques

The type or method of memory storage in a CBR system is one of the main contributors to the systems performance and resulting success. Ideally the case base needs to be stored in a highly manageable structure that allows for efficient search, retrieval and storage methods as mentioned in section 2.0. It is necessary to find equilibrium between the indices that define the access and retrieval of cases in memory, and storage methods that are able to preserve the semantic and often abstract complexities of cases.

Ideally one would like to maximise both aspects, so as to create a highly efficient, yet rich storage system. Retaining the abstractness of cases is fundamental to a CBR system, in order to attempt to replicate to as greater extent as possible the full semantics of a learning experience. Stripping an experience/lesson of its abstractness would be architecturally highly efficient but would result in a significantly diluted case base, in terms of usefulness in being able to provide or suggest a solution. The models used to store cases in memory are referred to as case memory models.

According to Watson and Marir (1994:p24), the two most influential models for the representation of case memory are the dynamic memory model of Schank and Kolodner (1982) and the category-exemplar model of Porter and Bareiss (1986). They are based on different views of how to organise cases in memory. Both models have their advantages and will be discussed further. The models can be distinguished as follows: (Kolodner, 1993:p108)

- One organising situations with similar activities, and the other
- Organising situations that are similar in the interactions between the goals and plans of people interacting in the situation. (Categories)

2.3.1.1 The Dynamic Memory Model

Referring to Section 2.1, Schank's research in the early 1980's was foundational in establishing the field of CBR. His paper on dynamic memory organisation (Schank, 1982) establishes the basis to the dynamic memory model. As mentioned in Section 2.1, this memory model was soon afterwards implemented in CYRUS (Kolodner, 1983) and many other subsequent CBR systems mentioned previously.

As supported by Schank (1982), the dynamic memory model makes use of Memory Organization Packets (MOPS) which organize situations (cases) whose activities are similar (Kolodner, 1993:p108). MOPS can be visualized as a form of container frame. MOPS are able to represent two distinct classes of events (Watson and Marir, 1994):

- Instances representing cases, events or objects, and
- Abstractions representing generalized versions of instances or of other abstractions.

A more advanced form of the MOP are Episodic Memory Organization Packets the (E-MOPS)). These are a development on the original MOP concept (Kolodner, 1983). E-MOPS are more widely referred to as Generalized Episodes (GE's) as suggested by Koton (1989).

The case memory, in the updated dynamic memory model consists of GE's organised in a hierarchical structure. A Generalized Episode is an indexing structure, which exists to allow the storage, matching and retrieval of cases. Cases that share similar properties are organised under the same GE. A GE in turn, has its own structure, consisting of three objects; cases, norms and indices (Aarnodt and Plaza, 1994):

- Cases are stored under a GE if they are considered similar, according to certain criteria.
- Norms represent the features that are considered common to the cases stored in the same GE.
- Indices are the features of the cases stored under the GE that allow the cases to be discriminated. An index is composed of an index name (reference) and an index value. An index can refer to another GE (usually one that is more specific) or a case.

The architecture of the case memory can thus be described as a discrimination network. The nodes in the discrimination network are either a Generalized Episode or its possible constituents, namely: cases, norms, index values or index names.

The Operation of Dynamic Memory

Dynamic memory is organised by a set of rules, which may differ by implementation, but the general concept is still the same, in that an attempt is made to allow for the fluent storage of new cases. The most likely operations to be performed on case memory are, search, retrieve and store. The retrieval operation encompasses and is dominated by the search procedure.

Retrieval

When a new problem (case) is encountered, the search process starts in an attempt to match the features of the new case to existing features in the discrimination network. This process begins at the root node and traversal continues down the discrimination network until one or more features of an existing GE matches the features of the new case. The new case is then further discriminated by its remaining features until the case with the closest match to the new case is found and retrieved.

Storage

The storage process is similar to the retrieval process. A new GE is created on the basis of the similarity of features of the new case to those of the existing case. For example: when an index name and an index value pair are matched. The two cases are then discriminated by placing them under different indices contained below the GE. If per chance that two GE's are created under the same index, they are not left to coexist, but are instead absorbed by the creation of a new GE.

The memory structure can thus be considered as dynamic, since similar components of two case descriptions are dynamically generalized into a new GE, with the cases being indexed under the GE by their dissimilarity in features (Aarnodt and Plaza, 1994).

One of the main problems of Dynamic Memory systems is that this method of dynamic storage and GE creation can result in rapid memory expansion. This occurs because of the growth of indices as case numbers increase (Watson and Marir, 1994). Thus it is practical in Dynamic Memory CBR systems to limit the choice of indices.

2.3.1.2 The Category-Exemplar model

The Category-Exemplar model is an alternative approach to case storage and management. This discussion on the Category-Exemplar model is based on the PROTOS system (Bareiss and Porter, 1986). PROTOS was developed to diagnose hearing disorders and is based on the preposition that "the real world should be defined extensionally" (Bareiss and Porter, 1986).

In the Category-Exemplar model cases are referred to as exemplars and are organized into categories according to their classification. Classification can broadly be defined as the process of determining what category cases can be organised under. Exemplar based classification is done by finding the exemplar most similar to the new case (exemplar)

and assigning the new case its classification (Kolodner, 1993). Further to this, a category's exemplars are organized according to their degree of prototypicality to the category. For example in the CBR system PROTOS (Bareiss and Porter, 1986), Bird is linked by strong prototypicality links to Robin and Canary (Kolodner, 1993).

Kolodner (1993) identifies the two primary processes involved in classification as:

- Identifying or finding the exemplars to match, and
- The actual matching process

The case memory structure of the Category-Exemplar model consists of categories, cases, and semantic relations (Bareiss and Porter, 1986; Aarnodt and Plaza, 1994; Kolodner, 1993; Watson and Marir, 1994). A case contains defining features, which are described by a name and associated value. A case's membership to a category is determined by its features, which can each be individually assigned a weighting in determining the cases membership to the category. Semantic relations provide a mechanism of associating and connecting cases and categories. They play an important role in the classification process and allow for rich knowledge representation in the model.

According to Watson and Marir (1994), there are five different types of semantic relations:

- Associative links that point from case features (problem descriptors) to a case or category.
- Case links which point from categories to associated cases.
- Difference links which point from categories to neighbouring cases which are only marginally dissimilar in features. Difference links give this model an ability to make 'good' matches (Kolodner, 1993).
- Censor links which provide negative associations.
- Prototypicality links as mentioned earlier.

An example of the structure and relationships in the category-exemplar model is illustrated in Figure 2.3.

The Retrieval, Storage and Learning Processes

When a new problem (case) is encountered, it is organized by its prominent features. The new cases features are combined into a pointer to a case or category that matches its features best. If a category is directly identified, then the most prototypical cases are traversed and returned for analysis. The storage mechanism is again similar to the retrieval mechanism; an appropriate category as the storage domain is selected by finding a matching case and then appropriate feature indices are created. Difference links are established to any incorrectly matched or inappropriate cases identified previously.

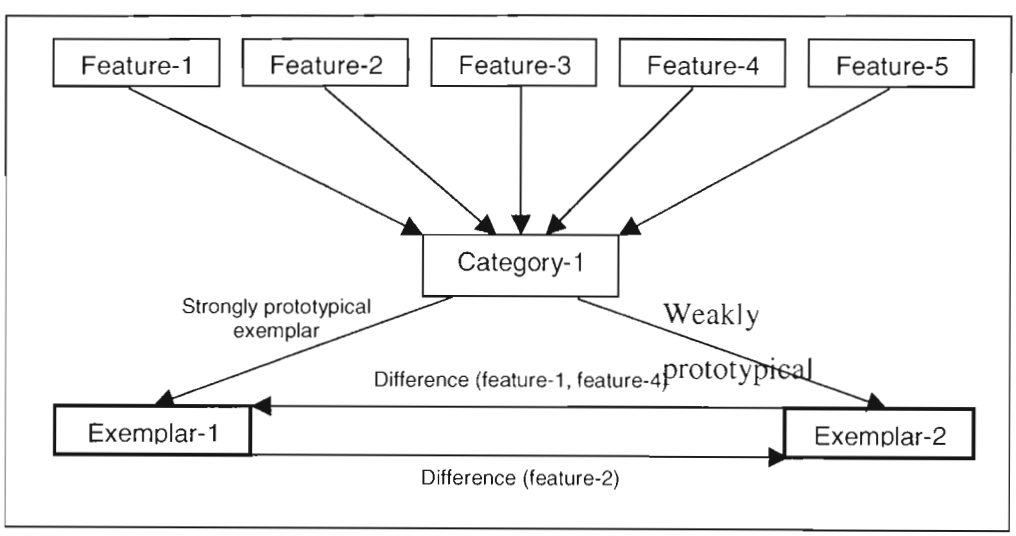

Figure 2.3 The structure of categories, features and exemplars (Aarnodt and Plaza, 1994).

Although the structure of the dynamic memory model is vastly different to the category exemplar model, both facilitate the process of CBR. Both models have been successfully implemented in many CBR systems mentioned in this chapter. However, the most adopted method of memory organization and management is Schank's dynamic memory model. This can be deduced from the fact that overwhelming majority of CBR systems

use the dynamic memory model in one variation or another. Aarnodt and Plaza (1994) stress the main problem with the category exemplar model; that it is extremely difficult to generalize cases to specific categories. The category exemplar model should not however be totally discounted for the reason that it has been implemented with great success in specific application areas. PROTOS (Bareiss and Porter, 1986) and CASEY (Koton, 1988) are examples of successful CBR systems implementing this method of memory organization and management. Kolodner (1993) states that exemplar based classification works when a model exists that can be used for matching and when it is easy to identify items in the real world with concepts in the concept world.

This chapter concludes with a discussion on the advantages of using CBR.

2.4 The Advantages of Case-Based Reasoning

When discussing the advantages of CBR, the literature tends to compare CBR with other knowledge-based systems such as rule-based reasoning. The reasons for this can be attributed to the field of application of each problem solving technique; both CBR and rule-based systems operate in the domain of solving real-world problems. In addition, knowledge-based systems in the form of rule-based reasoners are the most frequently implemented expert systems in the commercial environment (Turban and Aronson, 1998). Since it is not the purpose of this thesis to discuss the general field of knowledge-based systems, they are referred to in this discussion only for the purposes of indicating the advantages of CBR.

Knowledge-Based Systems have met with the following problems (Watson and Marir, 1994):

- Knowledge elicitation is a complex and difficult process. This problem is commonly referred to as the knowledge elicitation bottleneck.

- Implementation of Knowledge-Based Systems is a difficult, time-consuming process requiring special skills. This problem has been partially solved by the development of some sophisticated development shells.
- Once implemented, model-based Knowledge-Based Systems are often slow and are limited in their ability to access and manage large volumes of information.
- Knowledge-Based Systems are notoriously difficult to maintain. Rule-Based Systems require that in order for new knowledge to be gained, new rules need to be added to the system.

CBR is able to directly address the problems with traditional Knowledge-Based systems outlined above:

- CBR avoids the problems associated with knowledge elicitation and extracting and codifying the knowledge.
- CBR only makes use of real instances of problems. This implies that artificial algorithmic models are never used to model the problem domain. This avoids the risk associated with the potential inaccuracy of these models.
- CBR is able to deal with failed cases. These are cases for which an accurate prediction could not be made.
- CBR is particularly suited to weak theory domains (Kolodner, 1993). These are problem areas where the relation between cause and effect is not well understood (Delany et al, 2000).
- Users may be more willing to favour solutions proposed by a CBR system, since they are derived through a form of reasoning similar to human problem solving (Shepperd and Schofield, 1997). This is particularly important in situations where it is necessary that the user have confidence in the predictions made. Thus, CBR systems can be said to have high explanatory value (Mair et al, 1999).
- Rapid development and implementation of CBR systems is possible through the use of CBR shells such as REMIND (Inference Corportaion) and CBR Works (Tecc Inno).

- The learning mechanism in CBR allows for the immediate incorporation of new information into the case base. In addition to this failure driven knowledge acquisition protocols can be implemented in CBR to help improve the predictive capabilities of the CBR system.

It should be noted the advantages listed above are not intended to overshadow the use of other knowledge-based systems but rather to show the capabilities of CBR. Ketler (1993) states that CBR will not replace rule-based reasoning as an expert system technique but should be considered as an alternative complementary technique, which will allow for expert system implementations in domains not satisfactorily dealt with by other AI technologies. CBR has a number of qualities that make it particularly suited to domains in which other AI techniques are not particularly proficient. These are in particular: the ability to provide solutions to problems in weak theory domains (Delany et al, 2000); the ability to mimic the human thought process (Reisbeck and Schank, 1989) and the ability to seamlessly integrate new knowledge into the reasoning system.

The purpose of this chapter was to provide a theoretical and practical overview of the field of CBR. This was accomplished through: a discussion on the origins of CBR, a detailed description of the CBR process, a description and discussion on memory representation techniques and a summary of the advantages that CBR has to offer. CBR is able to provide an intuitive reasoning solution to problem areas, which have previously relied on experts to solve problems. CBR's inherent ability to automate human problem solving creates a niche application area for this technology. This will be discussed further in Chapter 4. The issue of case adaptation is probably one of the greatest challenges identified in this chapter. Watson and Marir (1994) emphasize the need for further research into this area. Furthermore, the available literature indicates that there is a need for further research into the limitations and application areas of CBR. This research intends to fulfil the need for such research by investigating the applicability of CBR to software cost estimation. This is a field of application that in the past has struggled to find effective solutions (Kemerer, 1987). Chapter 3 will provide a practical and theoretical background to the field of software cost estimation.

Chapter 3: Software Cost Estimation

This chapter will present an overview of the current state of research on software cost estimation techniques. Factors that are used as a framework to the discussion are:

- The nature of software development productivity.
- The classification of software cost estimation models.
- The methodologies used in cost estimation models.
- The operation of the various cost estimation techniques.

3.1 The Nature of Software Development Productivity

Software engineering practitioners have expressed concern over their inability to accurately estimate software costs (Kemerer, 1987). Accurate estimates are not only necessary for initial budgeting, where over and under estimates can be financially disastrous, but also for monitoring progress, scheduling resources and evaluating risk factors (Schofield, 1998). Financial consequences are not the only results of inaccurate software cost estimation; under estimated software costs can have a detrimental impact on the quality of the delivered software projects (Briand et al, 2000). Consequently, considerable research has been directed at gaining a better understanding of the software development process and developing more effective software cost estimation tools.

Boehm (1987) postulates that owing to the intangibility of the software development process, the costs of a software project cannot be mapped directly to quantitative measurements made to the actual development process. Instead, software costs are a function of productivity related to the outputs produced by the process and the inputs consumed by the process of software development.

Boehm (1987) defines productivity as a process:

$$\text{Pr}oductivity = \frac{Outputs\ produced\ by\ the\ process}{Inputs\ consumed\ by\ the\ process} \qquad (3.1)$$

Inputs consumed by the process generally comprise labor, hardware and supplies. Boehm (1987) states that the major problem in defining software productivity is defining the outputs. There have been two main measurements of outputs suggested by the literature. These are: Lines of Code (LOC) and functional complexity measurements. The COCOMO 81 model (Boehm, 1981) and Function Point Analysis (FPA) model (Albrecht, 1979) discussed below, are cost estimation techniques that use LOC and functional complexity measurement respectively.

A discussion on functional complexity and LOC models will be undertaken in the context of the models that use these measurements as their primary estimator. In order to discuss the various software cost estimation models that have been used, it is necessary to derive a descriptive classification for these models. This is undertaken in the next section.

3.2 Classification of Cost Estimation Models

There are a number of software cost estimation models available to the modern software engineer. Many of them, such as COCOMO 81 (Boehm, 1981) and Albrecht's FPA model (Albrecht, 1979) have gone through evolutionary improvement to take into account developments within the software industry over the years. These improvements have become necessary, as the way in which software is developed has evolved. Boehm (1995) states a number of factors that have affected the effectiveness of cost estimation models:

- Programming environments have changed considerably with most commercial software development now undertaken using fourth generation languages. The

immediate impact of this on software productivity is the impact of automatically generated code on programmer productivity.

- Object orientated programming has changed the way in which software projects are approached. In addition, object orientation has affected the way in which software maintenance is approached.

- The concept of design for reuse has had an impact on subsequent development projects and their maintenance in that reuse improves efficiency.

It is necessary to define a classification of effort estimation techniques in order to have an objective understanding of the functioning of the various effort estimation techniques. Many classifications of the available estimation techniques have been made in the literature (Conte et al, 1986; von Mayrhauser, 1990; Delany and Cunningham, 2000; Mukhopadhyay et al, 1992). The categorization of the estimation techniques is often subjective (owing to their complexity) and the deductive processes often not clear. It is probably for this reason that modern textbooks often avoid aligning with a particular classification

Von Mayrhauser (1990) proposes in her classification, three categories:

- Algorithmic Models
- Experiential and Statistical Models
- Subjective Experience Models

A more recent classification by Pfleeger et al (2001) also proposes three slightly different categories:

- Expert Judgment Models
- Algorithmic Models
- Machine-Learning Models

Both of these classifications use Algorithmic models as one of the categories. This can be considered a convenient category, where the majority of models are bundled together without distinction. The definition of an Algorithmic model according to Delany and Cunningham (2000) is one that uses mathematical formulae to predict effort and duration. This is a broad definition and could possibly include every estimation model other than subjective expert estimation. A notable departure from other classifications is Pfleeger's (2001) inclusion of the machine learning model type. This indicates the growing importance of machine learning techniques to the field of software cost estimation. Machine learning models are relevant to this research since this category also includes CBR.

Basili (1980) suggests a classification of software cost estimation models according to the form of the derivational equation that expresses the model and the level of detail of the model:

- Static or dynamic models
- Single variable or multi-variable models
- Theoretical or empirical models
- Micro or macro models

This classification is not representative of the underlying methodologies employed in each model and is thus not ideal to use as the basis for a summary on estimation methods. In his classification, Basili (1980) does not indicate the differences between descriptive and predictive models; Predictive models estimate the required effort before development. Models that fall into this category are COCOMO and Albrecht's FPA model. Descriptive models such as Halstead's Software Science approach (Halstead, 1977) are applied to existing software and are used to compare relative complexity and other comparative factors (Ghezzi et al, 1991).

Conte et al (1986) proposes the following, more extensive classification of cost estimation models:

- Historical – Experiential Models
- Statistical Models
- Theoretical Models
- Composite Models

Conte et al's classification or categorization has as the basis to its formulation, the method used in deriving each respective model. The abovementioned advancements in the sophistication of software have led to the development of a number of equally sophisticated estimation models, not in existence when this classification was made. It is for this reason that it is necessary to append this classification with a further category, namely machine-learning models as suggested by Pfleeger et al (2001). This new category of model makes use of modern machine-learning techniques such as CBR and Neural Networks to deal with the complexities of effort estimation in our modern software development environments.

Thus, for the purposes of this research the following classification will be used based on the above discussion:

- Historical – Experiential Models (Analogical)
- Statistical Models
- Theoretical Models
- Composite Models
- Machine-Learning Models

A discussion on this adapted classification follows with the intention of providing insight into each category, highlighting the most noteworthy estimation models in each. The models that have historically had the most impact on the field of software cost estimation

have been COCOMO (Boehm, 1981; Boehm et al, 1995) and Function Point Analysis (Albrecht, 1979). For this reason, these models will be discussed in more detail.

3.2.1 Historical-Experiential Models

These types of estimation model are often referred to as judgment or analogical models. Models of this type use an experience-based analysis and prediction mechanism for estimating software cost. This experience is retrieved locally from the involved developers (expert) and would usually be based on their personal opinions, augmented by internal historical data.

When more than one expert is involved, then a weighted average can be taken of their estimates. Other, more formal methods can be applied to improve this subjective form of estimation. One such method uses a probability distribution of experts' opinions (Pfleeger et al, 2001). This is done asking several experts to make three predictions: pessimistic(x), optimistic(y) and best guess(z). A normalized estimate is calculated using a formula of the form: $(x+4y+z)/6$.

An alternative way of formalizing an expert's opinion and giving the decision making process some structure can be done using the Delphi technique. The principles used in the Delphi technique were originally developed in the late 1940's to structure group communication processes to solve problems. This estimation technique enables several estimators to come to agreement by amalgamating their differing estimates into one (Pfleeger, 2001). Boehm (1981) refined this concept, so that it could be suitable for the task of software effort estimation. A number of discrete steps are followed until convergence is reached amongst the experts on an acceptable estimate.

The eight steps of the Delphi technique for software cost estimation as described in Pfleeger et al (2001, pg 611) are shown in the following table:

1. A group of experts receives the specification and an estimation form. 2. The experts discuss product and estimation issues. 3. The experts produce individual estimates. 4. The estimates are tabulated and returned to the experts. 5. An expert is made aware only of his or her own estimate; the sources of the remaining estimates remain anonymous. 6. The experts meet to discuss the results. 7. The estimates are revised. 8. The experts cycle through steps 1 to 7 until an acceptable degree of convergence is obtained.

Table 3.1 The eight steps of the Delphi technique.

Experiential methods of estimation can be applied in either a top-down or bottom-up fashion (as with most estimation models). Top-down estimation focuses on the system level, and takes into account system-level functions such as documentation, management and integration (Conte et al, 1986). Bottom-up estimation uses a componentized form of estimation, and considers the software project as a sum of system components. This implies that the final estimate made is the sum of the component estimates. The inherent advantage of bottom-up estimation is that the project is considered in more detail. This should theoretically provide a better estimate, but may result in the failure to consider factors such as management efficiency, testing and integration effort. This is where the top-down approach is more advantageous.

The most well known historical model is the (now itself historical) TRW Wolverton model (Wolverton, 1974). In it, costs are derived from a historical database of completed projects and their modularized costs. This model uses a Software Cost-Matrix to identify the cost involved in the six specified modules defined to be in the system under development, under the headings: Control, Input/Output, Pre/Post-Processor, Algorithm, Data Management and Time Critical. It should be noted that a system does not have to

be all inclusive of the above six modules. Furthermore, the complexity of each module is estimated on a six-point scale, under the headings OE (Old Easy), OH (Old Hard), NE (New Easy), NM (New-Medium) and NH (New Hard), (Pfleeger, 1998).

If C represents the cost matrix, then the elements of the matrix C_{ij} are given in Dollars per line of code. Where C_i is the module and C_j the difficulty. The overall system cost is given by Equation 3.2:

$$SystemCost = \sum_{all\ modules} C(k) \tag{3.2}$$

Where $C(k)$ is defined in Equation 3.3:

$$C(k) = S(k)C_{i(k),j(k)} \tag{3.3}$$

Where k is a module of size $S(k)$ in estimated LOC.

This cost matrix needs to be constantly updated to be representative of current costs. Conte et al (1986) state that this model introduces a level of objectivity, however they also state that there remains a great deal of subjectivity in the entire process. An example of a Wolverton Model Cost Matrix is given in Table 3.2 below:

	Difficulty					
Type of Module	**OE**	**OM**	**OH**	**NE**	**NM**	**NH**
Control	21	27	30	33	40	49
Input/Output	17	24	27	28	35	43
Pre/Post Processor	16	23	26	28	34	42
Algorithm	15	20	22	25	30	35
Data Management	24	31	35	37	46	57
Time-Critical	75	75	75	75	75	75

Table 3.2 Wolverton Cost Matrix (Pfleeger et al, 1998)

Owing to the transient nature of employment in the software development industry, much knowledge gained through experience is lost with rapid staff turnover cycles. It is therefore difficult to rely on developers for input as their experience base may vary from individual to individual. Experts can however incorporate into his/her estimation, unique knowledge of the organization, which could improve the accuracy of the estimate over other estimation methods. According to Pfleeger et al (2001), expert judgment methods are highly dependent on current data, often they neglect to take into account technical and other factors, and rely too heavily on an expert's ability to find similarities with past projects. This does not however condemn experiential and historical cost models but indicates the need to combine the positive aspects these models with more robust forms of estimation, one of which is CBR. Hence, their importance to the field of CBR for software cost estimation.

3.2.2 Statistically Based Models

Statistically based models, incorporate statistical methods such as regression analysis to produce a cost estimate. Conte et al (1986) divide this category into two, based on mathematical characteristics: The simplest statistical models are linear models, with the majority of statistical models being non-linear (see also Bailey and Basil, 1981; Walston and Felix, 1977). According to Conte et al (1986), linear models are not efficient at software cost estimation as compared to nonlinear models. Reasons for this include the fact that effort is an intrinsically non-linear entity and for this reason cannot be accurately predicted by linear regression techniques. The relative performance of linear to nonlinear models would explain the prevalence of non-linear models.

A generic linear model would be of the form:

$$E = c_0 + \sum_{i=1}^{n} c_i x_i \qquad (3.4)$$

where x_i are the factors affecting development effort (otherwise known as cost drivers). The number of factors affecting development effort are almost enumerable, with many of them having an insignificant effect on development effort. Different linear models include their own particular choices of cost drivers in the equation.

Much research into linear models was undertaken in the 1960's. In 1966 (Nelson, 1966) SDC (Systems Development Corporation) investigated the use of cost drivers to be included in a linear regression model using the least squares technique. Fourteen attributes were eventually identified as having a significant enough impact on total project effort to be included in the model. These are listed in Table 3.3 below.

	Cost Driver Attributes	Scale/ Value
x_1	Lack of Requirements	0-2
x_2	Stability of Design	0-3
x_3	Percent Math Instructions	%
x_4	Percent Input/Output Instructions	%
x_5	Number of Subprograms	Number
x_6	Programming Language	0-1
x_7	Business Application	0-1
x_8	Stand-alone Computer	0-1
x_9	First Program on computer	0-1
x_{10}	Concurrent Hardware Development	0-1
x_{11}	Random Access Device Used	0-1
x_{12}	Different Host, Target Hardware	0-1
x_{13}	Number of Personnel Trips	Number
x_{14}	Developed by Military Organization	0-1

Table 3.3. Fourteen Cost Driver Attributes according to Nelson (1966).

These attributes are the basis of the effort equation 3.5

$$\hat{E} = c_o + c_1 x_1 ... c_{14} x_{14} \qquad (3.5)$$

where $\hat{E}$ is the effort in person months.

According to Conte et al (1986), nonlinear models are generally of the form of Equation 3.6:

$$E = (a + bS^c)m(X) \tag{3.6}$$

where, S is the estimated size of the project in LOC, a, b and c are constants derived from regression techniques and $m(X)$ is an adjustment multiplier, dependant on the choice of cost driver attributes (X).

Most nonlinear models assume that the main factor affecting development cost is the size of the project. It is for this reason that size is usually estimated using LOC, which is subject to its own problems.

Walston and Felix (1977) derived a non-linear estimation technique based on the principles mentioned above. After analyzing IBM data from 60 projects, the following multipliers were arrived at. This is shown in Equation 3.7:

$$E = 5.25S^{0.91} \tag{3.7}$$

Where the effort E is a factor of system size S measured in LOC. This is the basic form of Equation 3.5, without having considered the adjustment multiplier factors. All prominent models (Bailey and Basili, 1981, Walston and Felix 1977, Boehm, 1981) produce an equation similar to that of Equation 3.6 and then provide qualifications of their specific choice of adjustment factors.

The Bailey and Basili model (1981) was intended to provide an organization specific modeling technique (Pfleeger, 2001; Conte et al, 1986). The basis for this model was the hypothesis that suggested that any constants used in an effort estimation model were significantly affected by the development environment and personnel at a given

organization. This modeling technique was based on NASA data from eighteen scientific projects. Most criticism of this model is targeted at the lack of variety in the data used in developing this technique (Conte et al, 1986). This implies that this model works well for scientific applications; especially ones developed using Fortran (as were the projects in the NASA database). The factors that Bailey and Basili (1981) considered in their model can be categorized into three areas affecting development effort (Table 3.4). These are the attributes that are used to adjust the effort of the Bailey and Basili effort equation.

Methodology (METH)	Complexity (CPLX)	Experience (EXP)
Tree charts	Customer interface complexity	Programmer qualifications
Top-down design	Application complexity	Programmer machine experience
Formal documentation	Program flow complexity	Programmer language experience
Chief programmer teams	External communication complexity	Programmer application experience
Formal test plans	Internal communication complexity	Team experience
Formal training	Database complexity	
Design formalisms	Customer-initiated program design changes	
Code Reading		
Unit Development folders		

Table 3.4 Factors affecting development effort (Bailey and Basili, 1981)

Contemporarily, statistical models have gone out of favor. This is primarily because of two factors: these models are highly calibrated to the data which was used to derive them and they are highly dependent on the size estimate of the software development (Pfleeger et al, 1998). Statistical models have made way for more sophisticated composite models such as COCOMO 81 (Boehm, 1981) and COCOMO II (Boehm et al, 1995), which acknowledge this problem and incorporate the use of a number of sizing techniques.

3.2.3 Theoretical Models

Theoretical methods try to overlay mathematical laws to the process of software development (Conte et al, 1986). Theoretical model describe relationships among product objectives, corporate assets and project objectives (Fairley, 1991). The most used theoretical model is Putman's Resource Allocation Model (Putman, 1978, 1984). This has since been incorporated into a commercial product known as SLIM (Software Life-cycle Methodology). Another popular approach is Halstead Software Science Model (Halstead, 1977). More recently Pillai and Nair (1997) used a Gamma Distribution equation as a basis to their theoretical model on software development effort and cost estimation.

The Rayleigh-Putman Model

Putman's Resource Allocation Model is based on the assumption that manpower utilization during program development follows a form of Rayleigh Curve (Putman, 1978). This is a negative exponential curve, which models the cumulative manpower distribution over time during a project.

The Putman model (Putman, 1976) has as its theoretical basis this hypothesis that the level of manpower effort required during the course of a software development project has a similar envelope (graphical) to that of the Rayleigh curve. The Rayleigh curve as applied to manpower estimation can be modeled by the following differential equation:

$$\dot{p} = 2adte^{-at^2} \qquad (3.8)$$

This equation expresses the Rayleigh expenditure rate $\dot{p}$ as a function of time. Here, t is elapsed time and a is a constant affecting the shape of the curve, where $a=1/(T^2)$, and where T is the time for manpower to peak. d is the cost scaling parameter (to fit the model to the project being modeled).

This observation was originally made by Norden (1958) who saw that the Rayleigh distribution curve provided a good approximation of the manpower required during the course of hardware development. This came from studying a number of hardware development projects at IBM. Putman pursued his initial hypothesis by studying military software projects to ascertain how his theory can be applied in describing the software lifecycle. From this research came a number of important observations:

- The Rayleigh curve reaches its maximum value, T, usually corresponding to the time of system testing and product release.
- The area under the Rayleigh curve at any time interval represents the total effort expended up to that interval.
- 40% of the area under a Rayleigh curve is to the left of T, and 60% is to the right. This is a reasonable model of how effort is distributed between the development and maintenance lifecycles.

Simply implementing the Rayleigh curve for any software project would not result in an accurate estimation, since software projects have a number of factors affecting their progress. It is for this reason that Putman introduced what he calls a technology factor C. This constant combines the effect of using tools, languages, methodology, quality assurance procedures and standards (note this is not an exhaustive listing). These factors can be said to implicitly affect productivity and are usually specific to the environment of the project. This constant can determined based on historical data or can be chosen from the twenty possible values suggested by Putman (Putman, 1978). This enables the developer to calibrate his project accordingly. To determine the constant from historical data, a least squares regression technique would normally be applied.

Problems with the Rayleigh-Putman approximation model can also be generalized to other approximation methods that use the Rayleigh/Norden equation as their basis (Conte et al, 1986; Fairley, 1991):

- These types of models appear to be significantly more accurate for larger software projects. Small to medium sized projects tend to be overestimated.
- The effect of schedule compression on a software project is radically over-exaggerated.
- The model provides little accuracy at the initial and end stages of software development, since most projects begin with a non-zero manpower level. This also applies to the maintenance phase.
- Fitting a Rayleigh curve to a given dataset through regular non-linear regression techniques is a complicated process that requires the aid of sophisticated statistical tools.

It must be noted that the Putman model does not utilize the Rayleigh curve as an estimate, instead the model attempts to fit manpower data to the curve. The use of the Rayleigh distribution as a model for fitting manpower data was an observation made by Putman for which he provides no theoretical reasoning (Fairley, 1991).

The Halstead Software Science Methodology

Halstead's Software Science method (Halstead, 1977) is a unique approach to predicting software effort in the early stages of software development (Conte et al, 1986). Halstead's Software Science approach has its origins in information theory and attempts to predict the qualities of a program before inception. Halstead attempts to provide a formal definition for qualitative, empirical, and subjective software attributes like ease of understanding, level of abstraction, based on low level quantitative software attributes like number of operators and operands appearing in a program (Ghezzi et al, 1991). These are referred to as Halstead metrics and are defined as follows:

- η_1 is the number of unique, distinct operators appearing in the program,
- η_2 is the number of unique, distinct operands appearing in the program,
- N_1 is the total number of occurrences of operators in the program,
- N_2 is the total number of occurrences of operands in the program.

The precise definition of these terms is said to subjective and is the source of much criticism of the Halstead method (Ghezzi et al, 1991). Halstead makes use of set theory and combines these features to produce effort estimation. Halstead's effort estimation is defined as the number of mental discriminations required to implement a program. This is a unique approach (considering the other approaches considered in this chapter). The defining effort equation is as follows in Equation 3.9:

$$E = V/L \tag{3.9}$$

where V is the program volume defined by Equation 3.10

$$V = N \log_2 \eta \tag{3.10}$$

V is intuitively the minimum number of bits required to code the program in question (V is referred to as the potential volume),
And,

$$N = \log_2(\eta_1 \eta_1 \circ \eta_2 \eta_2) \tag{3.11}$$

And,

$$L = V^*/V \tag{3.12}$$

where,

$$V^* = (2 + \eta_2).\log_2(2 + \eta_2) \tag{3.13}$$

V^* is referred to as potential volume and L is an attempt to quantitatively gauge the level of abstraction of the formulation of an algorithm. An assumption is that V^* is the most abstract possible formulation.

Ghezzi et al (1991) state that for small programs, effort E correlates with the effort needed for maintenance. As previously mentioned, Halstead's approach is intuitive to the point that it is attractive, yet the subjective nature of many of his assumptions and definitions leads to the questioning of his methods real world suitability and application. This makes it difficult to integrate with software development methodology. Fenton and Pfleeger (1997) and Maxwell et al (1996) have also directed much criticism towards the use of Halstead metrics. Maxwell et al (1996) state that they are based on questionable assumptions, which do not properly transform into accurate predictions.

3.2.4 Composite Models

Composite models incorporate a combination of analytic equations, statistical data fitting (linear or nonlinear) and expert judgment (Conte et al, 1986). Composite models can intuitively be partitioned into two distinct types according to the underlying metric used in estimation: Those that use Lines of Code (LOC) as a basis for estimation and those that use function points.

The most widely used composite model using LOC is the COCOMO model by Boehm (1981), more specifically the intermediate form of COCOMO is the most popular. A newer version of COCOMO, known as COCOMO II was more recently developed (Boehm et al, 1995) and takes into account the widely published criticisms of the 1981 COCOMO model.

Models that use function points as the basis of their estimation capability are based on the original Function Point model suggested by Albrecht (1979). The description of these

two models will be the focus of this sub-section. To avoid confusion where necessary, the original COCOMO model shall be referred to as COCOMO 81 as in Boehm (2000).

3.2.4.1 Models that Use Lines of Code as the Primary Estimator

The two most prominent models that use Lines of Code (LOC) as their primary estimator are COCOMO 81 (Boehm, 1981) and COCOMO II (Boehm et al, 1995). These two models are discussed in detail in the following sections.

COCOMO 81

COCOMO 81 (Constructive Cost Model) is an algorithmic cost estimation technique developed by Barry Boehm in 1981 whilst working for TRW. COCOMO 81 is a well documented and the most widely implemented of the cost estimation techniques described (von Mayrhauser, 1990). Knowledge of the operation of COCOMO 81 gives insight into the general operation of most algorithmic cost models. COCOMO 81 was developed by Boehm with the data collected from 63 software development projects. COCOMO was intended as an easy-to-understand model that predicts the duration of and effort required for a project, based on inputs relating to the size of the systems and a number of cost drivers, which Boehm identified as affecting productivity (Kemerer, 1987)

There are three variations of implementation within COCOMO 81, increasing in complexity and level of detail (Ghezzi et al, 1991). These are: The Basic, Intermediate and Detailed COCOMO. Each variation shows an increasing technical complexity used in estimating costs by applying cost drivers that adjust and improve the quality of estimation. Thus each more advanced level of estimation requires that more information is known about the software project than the previous level:

- The basic COCOMO is suitable for software projects where the project requirements are vague. The equation used is of a basic form and doesn't include the use of cost drivers.

- The intermediate COCOMO is suitable for software projects where the requirements are well understood. Here cost drivers are used to improve the accuracy of estimation. These cost drivers were selected by Boehm as the factors that he considered most influential on the productivity (Boehm, 1981).

- The detailed COCOMO model is very similar to the Intermediate COCOMO model, except that it is applied after the completion of each phase of a project. This allows for a work in progress perspective of a software project. Boehm (1981) identified 4 distinct project phases: Product Design, Detailed Design, Coding/Testing and Integration/Testing. After each of these phases the COCOMO model would be applied (taking into account additional cost drivers).

Before an explanation of the mechanics of the COCOMO model is given, it is important to have an idea of the bounds of the environment within the COCOMO model operates. For the COCOMO model to be effective, a number of assumptions are made: (Boehm, 1981):

- The model is more effective if the software projects are within the limits of 2K (thousand) to 32 KDSI (Thousands of Delivered Source Instructions).

- The software project domain has to be familiar to the developers.

- Software requirements are stable and the customer and the software developer will manage the project jointly (Sommerville, 1989). This serves to enclose and protect the development environment from being influenced by other external factors which could disturb the model.

- Effort multipliers are used to modify the above assumptions.

- The scope of the estimates take into account the following activities: Design through acceptance testing, the cost of documentation and reviews and the cost of the project manager and program librarian. The effort estimators exclude the cost of planning and analysis and of minor employees.

- Systematic software engineering techniques are used throughout the development process. This would include such activities as incremental documentation and careful definition and validation of architectural design.

COCOMO seeks to identify variables that would most significantly impact on productivity. The COCOMO model considers extensive factors regarding the type of project, implementation requirements and personnel constraints. These are considered in the following general equation that COCOMO uses for estimation of development effort:

$$E = aS^b m(X) \tag{3.14}$$

In the above formula, a and b are constants, which are intelligently selected according to the level of estimation (basic, intermediate or detailed) and category of software project (organic, semi-detached, embedded). The primary factor affecting this effort estimation formula is size, S. Size (S) is measured in KDSI (Thousands of Delivered Source Instructions). $m(X)$ is a cost multiplier and is a function of 15 other factors (cost drivers). Effort, E, is measured in Person Months. A Person Month is defined as the amount of time one person spends on working on the software development project for one month (Boehm et al, 2000). This is equivalent to 152 hours per Person Month. These further 15 considerations can be said to 'fine-tune' the predictive capability of COCOMO by considering such qualitative attributes as personal ability and experience (among others).

The necessity to identify possible contributing factors to software productivity came after Boehm found that the Basic COCOMO model was not accurate in predicting effort. He

found that the Basic COCOMO model predicted effort accurately on sixty percent of the time in his 63-project database (Kemerer, 1987).

The cost drivers identified by Boehm to enhance the COCOMO model are shown in Table 3.6 adapted from Boehm (1981) and von Maryhauser (1990). Here factors influencing total project effort are grouped into three attributes; Product, Process and Resource attributes. This grouping is not essential to the functioning of the model but is useful for illustrative purposes. The basic COCOMO model does not consider these cost drivers but only considers program size as the primary estimator.

Product attributes	
Required software reliability	
Size of application database	
Product complexity	
Process attributes	
Application of software engineering methods	
Use of software tools	
Required development schedule	
Resource attributes	
Computer attributes	**Personnel attributes**
Performance requirements	Analyst capability
Memory/Storage constraints	Applications experience
Virtual machine volatility	Programming capability
Turnaround time	Language experience
	Virtual machine experience

Table 3.5 Cost Drivers for the COCOMO model (Boehm, 1981; von Mayrhauser, 1990)

The three categories of software projects, namely Organic, Embedded and Semi-detached are defined as follows:

- Organic software projects are software projects of relatively small size. They require little innovation, are usually developed in-house and often don't have fixed deadlines constraining them.

- Embedded software projects are relatively large, require a much greater degree of innovation, have strict deadlines, and are generally more complex.

- Semi-detached projects fill the divide between the subjective definitions of Organic and Embedded projects.

The following Table 3.6 illustrates the effect that project category and estimation level have on constants *a* and *b* of the COCOMO 81 effort estimation formula.

Model	Basic		Intermediate	
	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

Table 3.6 The effect of project category and estimation level on constants *a* and *b* (Conte et al, 1986).

Total project development time *T* (months) can be determined as an adjusted product of Effort *E* (in PM – Person Months) as indicated in Equation 3.15. As previously mentioned, Person Months are defined as the amount of time one person spends on working on the software development project for one month (Boehm et al, 2000).

$$T = aE^b \tag{3.15}$$

where a and *b* are constants depending on development level, with *c*=2.5 for organic, semi-detached and embedded systems in the basic COCOMO model. Sommerville (1989) notes that the time required to complete a project is a function of the total effort required for the project and not a function of the number of software engineers working on the project. This confirms the idea that adding more people to a software project which is behind schedule is unlikely to get the project back on schedule.

COCOMO 81 has been much criticized in the literature. One of the major criticisms of COCOMO 81 is that it does not account for the reuse of existing code, although this can be considered a criticism of the majority of estimation models. A criticism directed at the LOC based models in general is that they require estimation before development, when accurate LOC estimates may only be available until the design phase is complete (Mukhopadhyay et al, 1992). The difference in programming languages greatly affects the LOC count, which is the basis of COCOMO estimation model. This implies that that the model needs to be recalibrated to suit the development environment (Jones, 1986). These factors have led to the development of an improved estimation model, COCOMO II (Boehm et al, 1995).

COCOMO II

COCOMO II builds on the success of the original COCOMO 81 model. COCOMO II is evidently immensely more sophisticated than COCOMO 81 and incorporates the use of function points. COCOMO II makes better use of cost drivers by providing less qualitative definitions for the cost drivers.

COCOMO 81 was developed in an era where there were preferred software life-cycles such as the Waterfall Model (Boehm et al, 2000). Software development models have changed considerably since then and need to take into account design for reuse concepts and object-orientated componentized software development. COCOMO 81 is inflexible in these regards. The newer COCOMO II model allows for the software engineer to adjust the cost-estimating model as more is known about the project during development (Boehm et al., 1995). With COCOMO II, there is an acknowledged realization that source lines of code (SLOC) are not an accurate basis for early effort estimation (Pfleeger et al, 2001). This had been the major source of criticism for the COCOMO 81 model (Jones, 1991). This knowledge is incorporated in the incremental development model of COCOMO II and for that reason allows for the term 'hybrid model' to be used in describing it (Boehm et al., 1995).

COCOMO II separates the development process into three major phases:

- In the initial phase (Phase I), prototypes are developed to resolve issues considered high-risk. These issues under consideration include (among others): user interfaces, performance and software and system interaction. At this phase quantitative size estimation methods using SLOC are notoriously inaccurate as little is known about the project. For this reason COCOMO II uses object points as a size estimate. Object points were originally proposed by Banker, Kauffman and Kumar, 1994. This method of size estimation uses high level generators to estimate size. These generators can include: number of server data tables, number of client data tables and percentage of screens and reports used in past projects.

- Phase II of the project considers alternative architectures and concepts of system operation are explored. At this phase a firm decision has been made to proceed with the project in question. Considerably more is known about the project at this phase and Function Points are used for size estimation. The cost drivers used here are of a lesser set than those used to derive an estimation in phase III (Pfleeger et al, 2001).

- Phase III is the phase at which development has begun. Here size estimation can be done in terms of Function Points or LOC. Pfleeger et al (2001) state that at this phase cost drivers can be estimated with some degree of comfort.

The basic form of COCOMO II has some similarity to its first version and is of the form (Equation 3.16):

$$E = bS^{c}m(X) \qquad (3.16)$$

Where as before, the size estimate bS^c is multiplied by $m(X)$ which is a function of the cost drivers (otherwise known as effort multipliers). The constant b approximates a productivity constant in PM/KSLOC (Person Months per Thousand Source Lines of Code) for the case where c=1.0. The exponent c is an aggregation of five scale factors that account for economies or diseconomies of scale encountered for software projects of different sizes (Banker et al 1994). Economies of scale follow the principle that as the project size increases, so does the productivity. The size, S is in KSLOC, which can also be estimated using UFP's (Unadjusted Function Points), which are then converted into SLOC. An overview of the scale factor c is presented in Table 3.7 below.

Exponent c	Relative Level of Economies of Scale	Circumstance
c<1.0	Project exhibits economies of scale	Project tools can help attain economies of scale.
c=1.0	Economies and diseconomies of scale are equal, resulting in a linear model	Usually for small software projects
c>1.0	Project exhibits diseconomies of scale	Overheads have become too large.

Table 3.7 The definition of the scale factor c (Boehm, 2001).

Boehm (2000) reiterates the necessity to use cost divers, since they uniquely capture the characteristics of the software development that affect the effort to complete the project. The cost drivers in COCOMO II have qualitative ratings on the impact of that driver on development effort. Ratings are on a seven-point scale ranging from Extra Low to Extra High. Each qualitative rating then has its associated effort multiplier, which allows the software engineers qualitative assessment to be used quantitatively in the model equations. It is important to note that a major advancement in the use in cost drivers was made with COCOMO II. The way in which cost drivers are used takes into account componentized project development, since most cost drivers apply to individual project components.

Boehm (2000) states that Size is still the most important input into the COCOMO II model (similarly to COCOMO 81). However, the theory behind the exponent adjustment

factor (discussed above) has advanced considerably since COCOMO 81 and is more representative of a real development situation. The cost drivers used at each stage of the COCOMO II estimation model are summarized in the Table 3.8.

Model Aspect	Stage 1 - Application Composition	Stage 2 - Early Design	Stage 3 -Post architecture
Size	Application points	Function Points	Function Points and language or SLOC
Reuse	Implicit in model	Equivalent SLOC as function of other variables	Equivalent SLOC as function of other variables
Requirements change	Implicit in model	% Change expressed as a cost factor	% Change expressed as a cost factor
Maintenance	Application, Point, Annual, Change, Traffic	Function of ACT, software understanding, unfamiliarity	Function of ACT, software understanding, unfamiliarity
Scale, c in nominal effort equation	1	0.91 to 1.23, depending on precendent, conformity, early architecture, risk resolution, team cohesion, and SEI process maturity	0.91 to 1.23, depending on precendent, conformity, early architecture, risk resolution, team cohesion, and SEI process maturity
Product cost drivers	None	Complexity, required reusability	Reliability, database size, documentation needs, required reuse and product complexity
Platform cost drivers	None	Platform difficulty	Execution time constraints, main storage constraints and virtual machine volatility
Personnel cost drivers	None	Personnel capability and experience	Analyst capability, applications experience, programmer capability, programmer experience, language and tool experience and personnel continuity
Project cost drivers	None	Required development schedule, development environment	Use of software tools, required development schedule and multisided development

Table 3.8. Cost Drivers used at each stage of COCOMO II (Pfleeger et al, 2001)

The COCOMO II model has the ability to be calibrated to a given project database. Boehm (2000) recommends that the model be calibrated to local data, so that it reflects local productivity and improves the models accuracy. Currently the COCOMO II development database is based on 162 software projects.

COCOMO II has the ability to take into account economies of scale, componentized development, software reuse, the local development environment and is able to refine the estimates at each project phase. This makes it a far more complete solution for estimation than COCOMO 81 and the other models discussed so far. The discussion on COCOMO II can proceed further, well beyond the scope of this chapter, since COCOMO II is not only intended as a mechanism for estimation but a complete planning methodology.

3.2.4.1 Models Based on Functional Complexity

Function Points are able to quantitatively measure a software project, by quantifying the information processing functionality associated with major external data or control input, output or file types (Boehm, 2000). There are several variations of this idea, discussed below.

Function Point Analysis as defined by Albrecht

The original function point methodology for effort estimation was developed by Allan Albrecht of IBM (1979). Albrecht determined in his hypothesis that software could be sized by evaluating the external transactions processed by an application or subsystem and the databases that were used (Herron, 2000). Thus, Function Point Analysis (FPA) measures software size by quantifying the functionality the software provides to the user based primarily on logical design. Albrecht's reasons for proposing function points as a measure of system size are stated as follows (Albrecht and Gaffney, 1983):

- The measure isolates the intrinsic size of the system from the environmental factors, facilitating the study of factors that influence productivity.

- The measure is based on the users external view of the system and is technology independent.

- The measure can be determined early in the development cycle, which enable function point counts to be used in the estimation process.

- Function points can be understood and evaluated by non-technical users.

Symons (1985) suggests that an implied aim was to develop a method which has an acceptably low measurement overhead. In its initial form Albrecht's (1979) function point model had as the basis to its estimation, four data requirements for input:

- Number of inputs
- Number of outputs
- Number of inquiries
- Number of logical master files

Albrecht's original FPA model was refined by Albrecht and Gaffney (1983) and has since been developed by the International Function Point Users Group (IFPUG). This IFPUG function point model forms the basis to this discussion on Function Point Analysis. There are many other forms of function point counting available to the software engineer. One of the models that makes the most significant methodological departure from Albrecht FPA is Mark II FPA developed by Symons (1985, 1991). Most other funtion point models use similar methodologies and will therefore not be discussed in this section. Mark II FPA is discussed in greater detail below.

FPA Mark II

An alternative function point model was proposed by Symons in 1985 and officially implemented in 1991. His model, known as Mark II Function Point Analysis (FPA) was aimed at addressing the shortcomings of Albrecht's FPA model that he identified in his research (Symons, 1985). These two methods measure subtly but significantly different sizes of the software product and therefore the work-output of the processes of developing, maintaining and enhancing the software product (Symons, 1991).

Symons (1985, 1991) identified that the most significant shortcoming in Albrecht FPA was that the classification of all system component types as simple, average or complex limited the accuracy of the measure in larger software developments. Symons (1991) states that Mark II FPA has a finer granularity and is a continuous measure, while Albrecht FPA limits component size once a threshold is reached.

In Mark II FPA, The Unadjusted Function Point count (UFP) is calculated through the use of the following formula:

$$UFP = N_i W_i + N_e W_e + N_o W_o \tag{3.17}$$

Where:

N_i = the number of input data element types

W_i = weight of an input data element type

N_e = the number of entity-type references

W_e = weight of an entity-type reference

N_o = the number of output data element types

W_o = weight of an output data element type

And N_i, N_e and N_o are each summed over all transaction types.

Symons (1991) defines transaction types as: logical transactions, inputs, outputs, data elements and entities. This count is then refined further by considering 19 other

characteristics in addition to any particular characteristics the user wishes to include. This is a similar system to the model developed by the International Function Point Users Group (IFPUG) method described in the next section.

Rask et al (1992) compared both techniques (FPA and Mark II FPA) by simulation, deriving all function types and counts automatically from dataflow diagrams and entity relationship diagrams. In general, they obtained good correlations between the models, with Mark II FPA arriving at a higher UFP count as the system size increases. However, the correlation was higher when the system sizes were smaller. Low and Jeffery (1990) analysed FPA counts as an effective measure of system size. Dolado (1997) summarized their findings as follows:

- Function points appear to be a more consistent a priori measure (early in the development of the software project) of system size than Lines of Code (LOC).

- There are significant differences in computing function point counts among different analysts and organizations.

- Experience in both software development and in function point counting was an important factor in obtaining a representative function point count.

The IFPUG function point counting procedure is discussed below.

Function Point Analysis Model Developed by the International Function Point Users Group

The IFPUG (International Function Point Users Group) was formed in 1986 and assumed the responsibility of standardizing and promulgating the function point metric. IFPUG have advanced the original Albrecht model and IFPUG FPA considers five (rather than four) user function types defined in Table 3.9 as adapted from Boehm (2000). These function types are further classified by complexity level. These complexity levels are

used to determine a set of weights, which are then applied to their corresponding function point counts to determine the UFP (Unadjusted Function Point) count.

Function Point Type	Function Point Name	Code	Description
Transactional Functions	External Input	EI	Count each unique user data or user control input type that enters the external boundary of the software system being measured.
	External Output	EO	Count each unique user data or control output type that enters the external boundary of the software system being measured.
	Internal Logic File	ILF	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file that is generated, used, or maintained by the software system.
Data Functions	External Interface Files	EIF	Files passed or shared between software systems should be counted as external interface file types within each system.
	External Enquiry	EQ	Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

Table 3.9 Definitions of function point types (IFPUG , 2000).

The IFPUG function point counting procedure can be described as a three-stage process (adapted from Abran, 1996):

- Determine the UFP count by counting the function points as defined in Table 3.9. Function point types can be divided into two categories: data functions and transactional functions. The low, average and high complexity values of: external inputs, external outputs, external interface files, internal logic files and external inquiries are totaled using IFPUG matrices to determine the total Unadjusted Function Point (UFP) Count. The calculation process uses standard multiplicative weights as suggested by IFPUG. These weights are function point specific and are shown in Table 3.10. Equation 3.18 shows how the UFP count is calculated:

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{5} w_{ij} x_{ij} \qquad (3.18)$$

Where, with reference to Table 3.10, w_{ij} = weight for row i (function point name) and column j (low, average or high weight), and x_{ij} = actual value of function point count in cell i, j.

- Determine the Value Adjustment Factor (VAF) through counting the predefined General System Characteristics (GSC's) which assess the environment and processing complexity of the software application. There are fourteen application characteristics (GSC's), which are counted on a rating scale from 0 to 5. The fourteen characteristics are described in Table 3.11. The VAF is calculated using the following formula:

$$VAF = 0.65 + 0.01\sum_{i=1}^{14} GSC_i \tag{3.19}$$

Where VAF is the value adjustment factor and GSC_i is the degree of influence score for GSC i.

- Determine the adjusted function point count, using the UFP count and the VAF. The adjusted function point count is determined by Equation 3.20 below:

$$FP = UFP \times VAF \tag{3.20}$$

Where, FP is the adjusted function point count, UFP is the unadjusted function point count and VAF is the value adjustment factor.

Function Point Name	Low Count x Weight	Average Count x Weight	High Count x Weight
External Input	_ x 3	_ x 4	_ x 6
External Output	_ x 4	_ x 5	_ x 7
Internal Logic File	_ x 7	_ x 10	_ x 15
External Interface File	_ x 5	_ x 7	_ x 10
External Inquiry	_ x 3	_ x 4	_ x 6

Table 3.10 Multiplicative weights applied to the individual function point counts (Kemerer, 1993).

General System Characteristic	Description	Degree of Influence
Data Communication	The degree to which the application communicates directly with the processor	0 to 5
Distributed Data Processing	The degree to which the application transfers data among the components of the application	0 to 5
Performance	The degree to which response time and throughput performance considerations influenced the application development	0 to 5
Heavily Used Configuration	The degree to which computer resource restrictions influenced the development of the application	0 to 5
Transaction Rate	The degree to which the rate of business transactions influenced the development of the application	0 to 5
Online Data Entry	The degree to which data is entered through interactive transactions	0 to 5
End-User Efficiency	The degree of consideration for human factors and ease of use for the user of the application measured	0 to 5
Online Update	The degree to which internal logical files are updated online	0 to 5
Complex processing	The degree to which processing logic influenced the development of the application	0 to 5
Reusability	The degree to which the application and the code in the application have been specifically designed, developed and supported to be usable in other applications	0 to 5
Installation Ease	The degree to which conversion from previous environments influenced the development of the application	0 to 5
Operational Ease	The degree to which the application attends to operational aspects such as start-up, back-up and recovery processes	0 to 5
Multiple Sites	The degree to which the application has been developed for multiple locations and user organizations	0 to 5
Facilitate Change	The degree to which the application has been developed for easy modification of processing logic or data structure	0 to 5

Table 3.11 General system characteristics used in calculating the value adjustment factor (IFPUG manual version 4.2, 2000).

Concluding Remarks on Function Point Analysis

The continued well-being of function points is in the hands of the IFPUG group. It is the responsibility of this group to convey to software engineers the effectiveness and unique capabilities of function points as an estimation technique. Herron (2000) states that Function Point Analysis has been misused and misunderstood in the past. It is the responsibility of the IFPUG to educate software engineers as to the correct method of implementation. There are many other function-based metrics that have not been covered in this chapter. These include (among others) Demarco's Bang Metric (Demarco, 1982), Jones and Capers Feature Points (1991) and Boeing's 3D Function Points. These have not been covered since the principles are generally the same and the popularity of these alternate techniques have not been strong, leaving IFPUG FPA as the dominant method.

Boehm (2000) states that function points are useful estimators since they make use of information which is available in the early stages of the software lifecycle, unlike the LOC based models. For this reason Boehm included the use of function point counts in his COCOMO II model. Kemerer (1993) states that despite their wide use by researchers and software and their growing acceptance by software engineering practitioners, they are not without criticism. Kemerer (1993) describes the following concerns in using the function point method:

- The first concern revolves around the alleged low inter-rater reliability of the function point counts, that is, whether two individuals performing a function point count for the same system would generate the same result.
- The second related concern has developed in part owing to the popularity of the function point method. A number of researchers, consultants and organizations have developed variants on the original FPA method developed by Albrecht. Kemerer's (1993) concern over these variants is that function point counts using these methods often differ from the Albrecht FPA method. Jones (1986) compiled a list of 14 named variations and suggested that values obtained using these variations may differ by as much as 50%.

- Kemerer' (1993) third concern is that counting function points is a labor-intensive operation. In his research, Kemerer found that collecting data for 100 function points took one hour. However, there has been much research into developing Computer Aided Software Engineering (CASE) tools to automatically collect function point data from software projects.

Since function points are a measure of software functionality, they provide a vehicle to estimate the cost and resources required for software development (IFPUG, 2000). The functional size of a software project can be translated into required effort through determining the level of productivity within an organization (MacDonell, 1994). This is usually organization specific and requires that the productivity of the developers be quantified in terms of function points per person-hour.

The following section describes machine-learning techniques that have been developed for the purposes of software cost estimation. These are a recent advancement in the field of software cost estimation and have shown much promise in comparison to traditional cost estimation techniques (MacDonell, 1994).

3.2.5 Machine Learning Estimation Techniques

Machine learning techniques have successfully been applied to such complex problem areas as speech recognition (Sejnowski and Rosenberg, 1987), adaptive control (Narendra and Parthasarathy, 1987) and markup estimation in the construction industry (Hegazy and Moselhi, 1994) to mention only a few. Only recently have machine learning techniques been considered as an alternative way of predicting software effort (Schofield, 1998; Mair et al, 1999). There are a number of cost estimation techniques that are of the machine learning type. These are case-based techniques, artificial neural networks and rule induction, fuzzy systems, and regression trees. Of these techniques, it is only CBR and neural networks that are receiving serious attention from researchers (Schofield, 1998). These Artificial Intelligence techniques are vastly different from the traditional

methods of estimation as they attempt to produce informed/learned decisions from the data that is historically available. In a sense this can be paralleled to Historical Experiential models discussed above, the core difference being that the informed decision is now automated.

Carbonell (1990) divides machine learning into four major paradigms:

- Inductive learning
- Analytic learning
- Genetic algorithms
- Connectionist learning

According to Carbonell (1990), case-based estimation techniques fall under the analytic machine-learning paradigm while neural networks are of the connectionist-learning paradigm. These two most prominent techniques used for software effort estimation (Schofield, 1998), namely case-based and neural network methods will be considered in the discussion below.

3.2.5.1 Artificial Neural Networks Applied to Software Cost Estimation

The term neural network applies to a multitude of modeling techniques using different architectures (MacDonell and Gray, 1996). The most frequently used form of neural networks for effort estimation is feed-forward networks trained using the back-propagation algorithm. This form of neural networks will be used as the basis to the discussion of neural networks used in effort estimation, since a discussion on neural network architecture and algorithms falls beyond the scope of the research. Typically a feed-forward, back-propagated network would operate in three phases: feed-forward of the input error, back-propagation of the output error and adjustment of the network weights. Neural networks of this form have successfully been applied to effort estimation in a number of studies, all of which show favorable results (see: Finnie and Wittig

(1997), MacDonell and Gray (1996), Mair et al (1999)). These models were consistently able to outperform regression models and the results are good enough to rival results obtained from CBR models. The major issues that seem to shadow the use of neural networks in this domain are: lack of configurability, ease of use and explanatory value (Mair et al, 1999).

Artificial neural networks are nets of interconnected processing elements capable of learning the mappings that exists between input and output data (Dolado, 2000). What makes neural networks so promising for cost estimation is that they are able to represent a number of interconnected, independent units. This is ideal for representing the various activities involved in a software development project (Pfleeger et al, 2001). What makes an even more compelling argument for the use of neural nets in cost estimation is that the equation relating input and output variables can be left unspecified, allowing previously unknown relationships to be modeled (Dolado, 2000). This falls well beyond the scope of traditional estimating techniques where any unknown variables would make computation improbable.

A typical network created for the purposes of effort estimation would have as inputs: system size, several general system characteristics (GSC's) and the programming environment. The exclusive system output would be the estimated development time (Wittig and Finnie, 1997).

In a neural network, each activity is represented by a neuron (and graphically by a network node), with each activity having inputs and outputs. Figure 3.1 depicts the basic operation of a neuron making use of a step function for activation. Each neuron of the neural network has its associated software that performs an accounting of its inputs, computing a weighted sum as described as follows:

Each input is multiplied by the weight associated with that particular connection. The weighted inputs are then computed for that neuron. This value is then compared to the threshold value assigned to that neuron and the output is determined by this comparison.

This output can either be an excitory on inhibitory input to other neurons in the network. This process continues until the neural net produces a final cascaded output value (Pfleeger et al, 2001).

In a typical feed-forward neural network using the back-propagation learning algorithm, the difference between the actual and desired output is determined continuously for each set of inputs and corresponding set of outputs. A portion of this error is backward propagated through the network and used to adjust the weighting system so as to improve subsequent calculations. This process then continues until the network performance is optimized. According to Rumelhart et al (1986), classical back-propagation is a gradient method of optimization executed iteratively, with implicit bounds on the distance moved in the search direction in the weight space. This is achieved by incorporating a learning weight (gain) and the momentum term (damping factor) in the model.

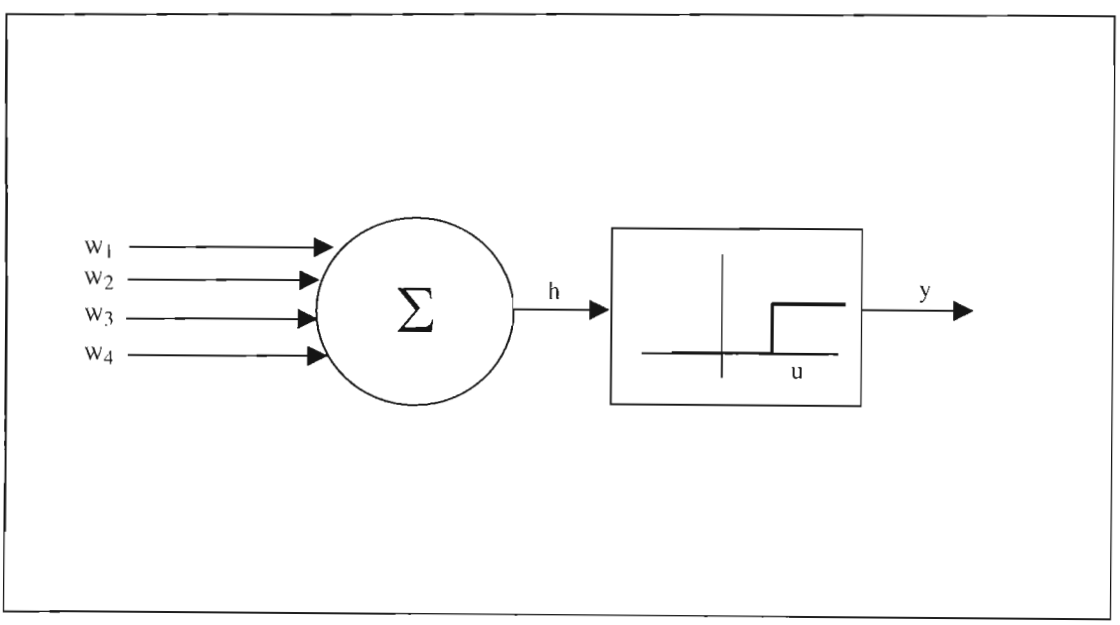

Figure 3.1 A McCulloch and Pitts Neuron (McCulloch and Pitts, 1943) adapted from Mair et al (1999).

According to Beale and Jackson (1991) this is the guiding principle behind the neurons ability to learn from previous mistakes. Thus, the error produced is used constructively

adjust the weights of the connections so that in the next incidence the level of error is improved.

Equation 3.21 summarizes the reactive behavior of an artificial neuron (equation and associated description taken from Schofield, 1998):

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right) \tag{3.21}$$

The operation of Equation 3.19 can be described as follows:
The neuron computes the weighted sum of its inputs, x_j, where $j=1,2,...n$. An output of 1 is generated if the sum is above a certain threshold u. Else an output of 0 is generated. $\theta()$ is a unit step function at 0 and w_j is the synapse weight associated with the jth input. u is considered as a weight i.e. $w_o=-u$ attached to the neuron with a constant input of $x_o=1$. Positive weights model excitory synapses, while negative weights model inhibitory ones. The activation function in Figure 3.1 is known as a Step function, however there are a number of functions that can be utilized such as Gaussian, Linear, Sigmoid and Tan*h*. Sigmoid functions are most frequently used in neural networks (Finnie and Wittig, 1997).

Two general types of neural networks exist: feed-forward and feedback networks. They differ architecturally, with feed-forward networks containing no loops in the network path, while feedback networks may have recursive loops in the network path. A feed-forward network is said to be of a static nature and has no memory of previous network states. A typical structure of a feed-forward neural network applied to effort estimation is shown in Figure 3.2.

The performance and operation of a neural network is reliant on the given architecture and parameter settings of the network (Dolado, 2000; Finnie and Wittig, 1997). The architecture of a network has a direct impact on performance criteria such as learning speed, accuracy and noise resistance. This makes configuration of a neural network

extremely difficult. There is no pre-defined method that is available to neural network developers that can suggest suitable parameter settings.

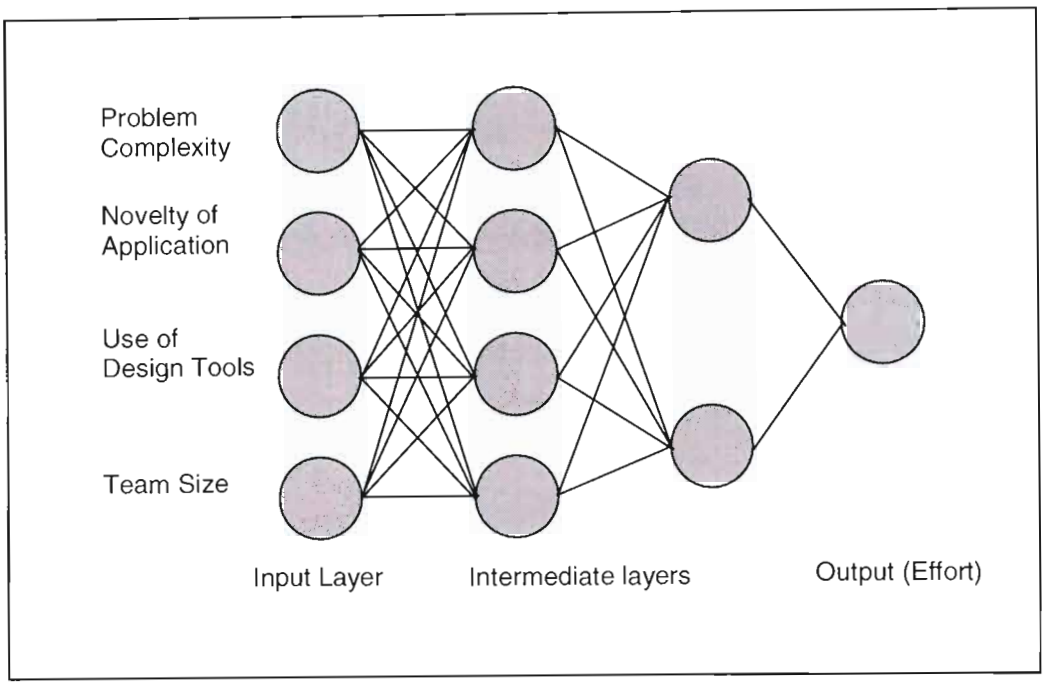

Figure 3.2 An example of a multi-layer perceptron (Rosenblatt, 1962) for software cost estimation (taken from Pfleeger et al, 1998)

Neural networks have to go through a 'training' process, where the relevant weights are refined. The implications of this are that an appropriate training set of data needs be fed into the network. This means that subsequent results will be greatly affected by the original training data (Finnie and Wittig, 1997). A disadvantage of this process is that once the network has been trained, it cannot be adjusted in any way. If the developer wished to adjust any parameters, the network would have to re-trained with training data.

As previously mentioned, Finnie and Wittig (1997) made use of a back-propagation multi-layer perceptron when they experimented with software effort prediction on two datasets. The datasets they used, were the Desharnais and Australian Metrics Association (ASMA) data sets. Both of these datasets were tested three times, so as to make sure of the confidence of the results. Finnie and Wittig found that the best results were

consistently obtained using a learning rate of 0.3, with a momentum coefficient of 0.6 on a network architecture with a single hidden (or intermediate) layer. In addition to this they concluded that the use of a Guassian and Sigmoid activation function resulted in the lowest prediction errors. The results obtained were impressive: for the Desharnais dataset an average MRE (Magnitude of Relative Error) of 29% was obtained, whilst for the ASMA datasets an MRE of 17% was obtained. From these datasets, ten projects were randomly chosen to be used as the test set wile the rest were used as training data.

Similarly to Finnie and Wittig (1997), Venkatachalam (1993), Jorgenson (1995), MacDonell and Gray (1996) and Srinivasan and fisher (1995) have all used back-propogation neural networks of similar configuration. All results obtained were promising, with the neural network consistently out-performing traditional algorithmic estimation techniques. Jorgenson (1995), MacDonell (1996) and Srinivasan and fisher (1995) did however question the robustness of this form of approach to effort estimation, since results were often inconsistent and required vast amounts of training data. Srinivasan and fisher (1995) indicated the difficulty in finding the best configuration of a neural network and suggested that in order to do this, th trianign data should be partitioned into smaller portions, so that more configurations can be tested with the test data.

Karunanithi (1992) attempted to make use of different network neural network architectures in their investigation. They made use of both a feed-forward and feed-back neural network using an algorithm known as cascade correlation. This algorithm incorporates in it, some aspects of the back-propagation algorithm. They reported success in that their model outperformed algorithmic estimation models.

Dolado (1999) developed a neural network for effort estimation in their investigation into effort estimation methods. He compared three different estimation techniques: Genetic programming, a regression model and neural networks. Dolado made use of a feed-forward neural network with a structure consisting of two layers and two neurons in the hidden layer. The primary activation function used was sigmoidal and the learning

algorithms employed were Levenberg-Marquardt and classical back-propagation. They conclude that the former algorithm has a performance advantage over the back-propagation algorithm. Using the test data, multiple runs of the neural network were performed in order to optimize the weights from those initially randomly chosen. Dolado's reaction to his results was far less optimistic than those of the other researchers mentioned above. He concludes that his neural network system was unable to attain acceptable results in that the neural network was unable to consistently outperform classical regression measures. However, in some cases the neural network was able to outperform the other models. Dolado concludes that neural network models have the potential to perform as well as other estimation methods but without any significant benefit.

The results in the literature regarding the performance of neural networks as applied to effort estimation are in general positive, indicating that they have potential in this domain. However, most of the researchers mention the difficulty in finding an appropriate configuration and consistent results.

MacDonell and Gray (1996), state that an obvious disadvantage of neural networks is their "black box" nature, where inputs and outputs are visible, but the underlying process of getting one from the other is hidden. This view is supported by Mair et al (1999) who state that neural network models are significantly lacking in explanatory value. In addition to this, neural networks also have operational limitations; as mentioned previously, once the learning phase of a neural network is complete, the analyst cannot manipulate the parameters set by that process (Dolado, 1999). MacDonell and Gray (1996) suggest as an alternative, the use of hybrid fuzzy neural networks, since they are able to provide the advantages of model-free estimation, non-linear mappings and good generalization capability. One of the greatest appeals for using neural networks in general, is that if there is any underlying relationship in the data being used, then a neural network should be able to find it (Dolado, 1999). Dolado makes an important statement regarding research in the field of effort estimation. He states that all results that have been achieved in the research are contingent on the datasets used. In all of the research

into estimation models, the datasets have consisted of data which he terms "simple data". This refers to the small number of datasets available to researchers such as the COCOMO 81 dataset, Albrechts dataset (1983) and the Kemerer dataset (1987). It is generally accepted that in order to improve the estimation methods and attain a better quality of research it is necessary to have better quality data available to the researchers.

3.2.5.2 Case-Based Reasoning as a Software Cost Estimation Technique

In an examination of the feasibility of a case-based reasoning model for software effort estimation, Mukhopadhyay and Vicinanza (1992) state that traditional algorithmic techniques fail to produce accurate software development effort estimates. This creates scope and gives credence for the development of CBR models in this problem domain. A discussion on CBR as an alternative cost estimation technique is presented in Chapter 4.

Chapter 4: Case-Based Reasoning Applied to Software Cost Estimation

This chapter presents an overview of the current state of research on applying CBR methods to cost estimation to demonstrate the applicability of CBR to software cost estimation. In addition to this, CBR development tools which are often used for developing CBR software cost estimation models are discussed in the context of this research.

4.1. The Applicability of CBR to Software Cost Estimation

Researchers have repeatedly stated that existing algorithmic models of software cost estimation fail to produce consistently accurate software development effort estimates (Kemerer, 1987; Mukhopadhyay et al, 1992; Kadoda et al, 2000). This has been extensively discussed in the literature, with many empirical evaluations of the various estimation techniques available (Kemerer, 1987, Mukhopadhyay et al, 1992). It is the lack of consistency and accuracy that has resulted in the ever-increasing interest in machine learning techniques as an alternative (Mair et al, 1999). Finnie and Wittig (1997), Mair et al (1999), Schofield and Shepperd (1995 and 1997) are some prominent researchers that have developed and tested various analogy-based software cost estimation models. Their research is discussed in Section 4.3.

CBR has been extremely successful in many areas of application, particularly in the fields of diagnosis, repair and adaptation, design and arbitration to name only a few (refer to Chapter 2: Table 2.1 for more examples). The fields of application are evidently diverse yet they have some commonality in that the problem domains are usually areas that are reliant on human expertise to provide solutions. Since CBR is a problem solving technique that relies on past experiences to suggest solutions (Riesbeck and Schank, 1989), it is highly suitable to these weak theory domains. Weak theory domains are those in which the relationships between cause and effect are not well understood. Software cost estimation is such a weak theory domain (Delany and Cunningham, 2000) and much

research has been conducted into the use of CBR in this area (Mukhopadhyay et al, 1992; Kadoda et al, 2000; Schofield and Shepperd, 1995,1997).

Boehm (1981) suggested the use of analogies as a possible cost estimation technique as far back as 1981. However, further research into this possibility was not considered until the late 1980's (Cowedroy and Jenkins, 1988). In the last decade a number of analogical and CBR cost estimation models have been developed. These have met with considerable success when compared to existing modeling techniques such as COCOMO 81 (Boehm, 1981) and Function point Analysis (Albrect, 1979) (Schofield and Shepperd, 1995,1997; Mukhopadhyay et al, 1992).

4.2. The Distinction Between Analogical and Case Based Reasoning.

Analogical reasoning and CBR are very closely related and it is for reasons of completeness that an appropriate distinction is drawn. The problem with a term like *analogy* is that it is overloaded with different meanings and no definition is precise. Aarnodt and Plaza (1994) state that analogy based reasoning is sometimes used as a synonym to case based reasoning and furthermore that case-based reasoning is often used as a generic term to describe systems that organize, retrieve, utilize and index the knowledge retained in past cases. They place CBR in the same domain as exemplar based reasoning, instance-based reasoning, memory based reasoning and analogy-based reasoning. Other researchers like Shepperd and Schofield (1997) state that estimation by analogy is a form of CBR. Serengul (2000) place CBR in the category of "hybrid learning" algorithms along with analogy and explanation-based learning.

Mukhopadhyay et al (1992) describe the distinction between CBR and analogy-based reasoning through the relative restrictions placed on the source and target domain of operation. They argue that analogical problem solving should in theory allow for the mapping of a source analog, which has a conceptually different domain to that of the target (new case/problem). However, in CBR the source analog is extracted from the

same general problem domain as the target. Furthermore, the source and target analog have the same structural representation. It can therefore be concluded that analogical reasoning and CBR operate similarly differing only in the parameters of operation. This is supported by Aarnodt and Plaza (1994), who use as a basis for this analysis, research that has been carried out in these fields. They state that analogy-based reasoning is the method used to solve new problems based on past cases from different domains, while case-based methods focus on indexing and matching strategies for single domain cases. They conclude by stating that research on analogical reasoning is a sub field concerned with mechanisms for identification and utilization of cross-domain analogies. It is focused on mapping the solution of an identified analogue to the present problem.

An overview of existing CBR based cost estimation techniques follows. Even though many of the systems mentioned in this chapter may be examples of some other learning approach like analogical reasoning, this does not mean that it is excluded from being a typical CBR application as well. This differentiation is not considered a critical factor in the discussion on the applicability of CBR to cost estimation and the terms analogical and case-based reasoning are used as per the authors/researchers statement without preference. The aim of the overview is to elaborate on some of the historical CBR systems and their success to demonstrate the applicability of CBR to the domain of software cost estimation.

4.3. Existing Analogical Reasoning Techniques Applied to Software Cost Estimation

There are only a few analogy-based effort estimation tools that have been developed (Shepperd and Schofield, 1995; Jeffery and Walkerden, 1999; Finne and Wittig, 1997; Mukhopadhyay et al, 1992, Briand et al, 2000). These have been exclusively for academic research purposes and thus far to the best knowledge of the author, none have been implemented commercially. As detailed in Chapter 2, CBR is a machine learning paradigm that has been in existence for approximately two decades, yet only in the last decade has there been any significant research into CBR's application to software cost estimation. More precisely, most of this research was undertaken in the latter half of the

past decade. This is indicative of the maturity that CBR as a problem solving technique is attaining with the result that problems that more closely represent the human thought process are being attempted. Several of the reported results in this area will be discussed below.

Mukhopadhyay et al (1992) were the first to investigate and implement CBR as a software cost estimation model. They developed a tool called ESTOR with which they performed an experimental investigation into the feasibility of using CBR for software cost estimation. ESTOR made use of function point and intermediate COCOMO metrics as inputs to case-base. These inputs were obtained from a previous study by Vincinanza, et al (1991) which augments the Kemerer (1987) dataset with additional projects. ESTOR selects an analogue (nearest matching case in the case-base) for the new software project by calculating the Euclidean distance between completed projects and the new project and then selecting the nearest neighbor. In order to adjust the estimate suggested by ESTOR, a number of production rules are used. A conflict set of rules is created for non-corresponding attributes, which are accordingly adjusted to resolve the conflict (Mukhopadhyay et al, 1992).

ESTOR's predictive capabilities were measured against the predictions made by an expert in the software engineering field and predictions made by using COCOMO (Boehm, 1981) and Function Point Analysis (Albrecht, 1979). In the respect that Mukhopadhyay et al (1992) compare the predictions made by ESTOR to an expert is methodological departure from more current research, which tends to compare the accuracy of the developed analogy-based estimation model to algorithmic models. Mukhopadyay et al (1992) found that for the 15 projects for which ESTOR and the expert had to suggest solutions; the Mean Measure of Relative Error (MMRE) for the expert was 31% and for ESTOR, 51%. MMRE is a measure of absolute relative accuracy, which will be discussed in detail in Chapter 5. MMRE can briefly be defined as the absolute percentage error of the prediction to the actual effort. In conclusion to their research, Mukhopadhyay et al (1992) state that ESTOR produced significantly more accurate estimates than Function Point Analysis and COCOMO. These results are indicated in

Table 2.1. These positive results give credence to the use of CBR as a cost estimation technique. Kadoda et al (2000) do however criticize the ESTOR CBR model, stating that a disadvantage of the ESTOR CBR model is that it is very specific to the particular dataset since the rules and similarity measures are defined in terms of the features available.

Table 4.1 Relative accuracy of ESTOR to other models (Mukhopadhyay et al, 1992)

Method	MMRE %
Expert	30.72
ESTOR	52.79
COCOMO	102.74
Function Point Analysis	618.99

The ANGEL (ANaloGy softwarE tooL) software is a more recent analogical cost estimation tool developed by Schofield and Shepperd (1995). Shepperd and Schofield have played a central role in promoting analogical reasoning as a solution to software cost estimation. Similarly to ESTOR, ANGEL identifies the most similar cases through using an algorithm that is based upon the minimization of Euclidean distance in n-dimensional space (Schofield and Shepperd, 1995). What makes ANGEL different from other analogical tools, is its dynamic ability to deal with differing datasets (Shepperd and Schofield, 1995). This means that the ANGEL tool can be easily adapted to the dataset used as the basis to estimation. Schofield and Shepperd (1995) prove this ability by testing it on three different datasets. Schofield and Shepperd (1995) do however point out that ANGEL is a prototype and is limited in the number of variables it can handle for determining the optimum combination of variables for finding analogies. The figure for the limitation number of variables is currently ten. This is a severe limitation, which would place restrictions on ANGEL's dynamic ability to use different datasets. In conclusion to their research (Schofield and Shepperd, 1995), it was determined that the analogy based reasoning approach outperforms the algorithmic approach against which they compared their results. In this particular case, this algorithmic technique was linear regression. A summary of Shepperd and Schofield's (1995) findings are indicated in Table 4.2.

Table 4.2 MMRE values for effort estimation (Shepperd and Schofield, 1995)

Dataset	ANGEL (MMRE %)	Regression (MMRE %)
Albrecht	62	90
Atkinson	39	99
Kemerer	62	106

Subsequent to the preliminary research by Shepperd and Schofield (1995), a number of papers have been published that make use of the ANGEL software tool (Briand et al, 2000; Jeffery and Walkerdon, 1999). The primary thrust of research by the developers of ANGEL (Shepperd and Schofield, 1995) has been to investigate the effects of different datasets on the accuracy of the ANGEL estimates (Shepperd et al, 1996; Shepperd and Schofield, 1997). A major criticism of ANGEL is that it does not provide a mechanism for case adaptation (Jeffery and Walkerden, 1999). Instead, Shepperd et al (1996) use the average effort value for two or three of the closest analogies suggested by ANGEL. Averaging is only likely to improve the estimate where the sizes of the closest projects straddle the size of the target project (new problem case) (Jeffery and Walkerden, 1999).

In a methodological departure to the research reviewed above, Briand et al (2000), Wieczorek (2001), Finnie and Wittig (1997) and Jeffery and Walkerden (1999) have developed analogy-based software cost estimation models using commercially available CBR tools. These tools facilitate the development of customizable CBR systems. CBR tools will be discussed in more detail in Section 4.4.

Jeffery and Walkerdon (1999) developed an analogical reasoning prototype tool called ACE (Analogical and Algorithmic Cost Estimator). As in ESTOR and ANGEL, ACE estimates effort for a new project by searching through a database of metrics for completed projects and selecting the completed projects it determines to be closest to the new project. They made use of a custom dataset, which they derived in earlier research (Jeffery and Stathis, 1999). Their findings are summarized in Table 4.3 below. The results they obtained indicated that ACE produced more accurate results than either ANGEL or the linear regression model.

Table 4.3 Mean Errors of the estimation methods used (Jeffery and Walkerden, 1999)

Estimation Method	MMRE %
ACE	55
Linear Regression	68
ANGEL	112

Finnie and Wittig (1997) developed a CBR software cost estimation model using the Remind CBR development tool (Cognitive Systems Inc.). To date they have shown the most positive results produced by a CBR software cost estimation system; achieving an MMRE figure of 36.2% on a large project dataset of approximately 300 software projects. Briand et al (2000) report more modest results for their analogy-based software cost estimation. They developed a CBR software cost estimation system using the CBR development tool CBR Works 4 (Tecc Inno). They achieved an MMRE of 74% from testing conducted over the entirety of the dataset.

Table 4.4 summarizes the results from all the researchers mentioned above. These results indicate that CBR is highly applicable to the problem domain of software cost estimation. This is validated by the individual researchers in comparing their models with various regression techniques and traditional algorithmic models such as COCOMO and Function Point Analysis. For example: Mukhopadhyay et al (1992) found that their CBR model significantly outperformed COCOMO and Function Point Analysis (see Table 4.1); Shepperd and Schofield (1997) demonstrated that regression models were unable to outperform ANGEL using most of the, then currently available datasets outlined in Table 4.4.

Table 4.4 shows that the best achieved MMRE results were those obtained by Finnie and Wittig (1997), with an MMRE of 36%. Their methodology in obtaining their results is outlined in Chapter 5. A significant observation that can be made from Table 4.4 is that Finnie and Wittig's research was conducted on a much larger dataset than those used by other researchers. They are the first researchers to have made use of such a large dataset. The most recent work was conducted by Briand et al (2000) who obtained an MMRE of

74% from their results. To summarize the results obtained by other researchers, an average MMRE figure over all the projects in Table 4.4. The average arrived at was 55.73%. Although averaging results is not methodologically prudent, it provides an indication of the results one can expect to obtain using CBR in software cost estimation.

Table 4.4 Summary of results obtained from research into analogy-based software cost estimation.

Analogical/CBR tool and Researchers	Year	Dataset name	No. of projects	MMRE %
ESTOR / Mukhopadhyay et al	1992	Kemerer	15	52.79
ANGEL / Shepperd and Schofield	1995	Albrecht	24	62.00
		Atkinson	21	39.00
		Kemerer	15	62.00
	1997	Desharnais 1	77	64.00
		Finnish	38	41.00
		Mermaid	28	78.00
ACE / Jeffery and Walkerden	1999	In house	19	55.00
Finnie and Wittig	1997	Desharnais 2	299	36.20
ANGEL / Mair et al	1999	Desharnais 1	77	49.00
Briand et al	2000/2001	European Space Agency	166	74.00
			Average MMRE %	**55.73**

Following the research reviewed above, the typical features and considerations for developers of analogy-based reasoning systems can be summarized as follows:

- Case structure.

The case structure of an analogy-based reasoning system is formulated from the data that is used as an input to the system. It is the developer's responsibility to identify appropriate features of the data for inclusion in the case-base of the system. Case structure has implications on the similarity and adaptation mechanisms used.

- Similarity measures.

Similarity measures are necessary in order to identify which cases are similar to the input case. The most popular form of determining similarity is through the minimization of Euclidean distance in n-dimensional space (as implemented by Shepperd and Schofield (1995), Briand et al (2000) and Wieczonek (2001)).

- Adaptation measures.

Case adaptation is currently an area of much discussion and speculation (Kolodner, 1993). Adaptation is necessary in order to place the problem case in the same context as the retrieved case(s). A linear extrapolation along the lines of some quantifiable measure of development project size is currently the most popular method (as implemented by Jeffery and Walkerden (1999) and Briand et al (2000)).

- Measures of prediction accuracy of the estimation model.

It is necessary for the researcher to be able to determine the accuracy of the predictions made by the developed system. Implementing a standardized measure of prediction accuracy allows for comparison with other analogy-based models (verification) and serves to validate the relative success of the model developed. Two measures have been applied consistently in all the research reviewed, these are: the calculation of the Mean Measure of Relative Accuracy (MMRE) and Prediction at Level (l). Both of these measures were proposed by Conte et al (1986) as appropriate measures of system accuracy. These measures and their derivations are discussed in detail in Chapter 5.

Developing a custom CBR model for software cost estimation like ESTOR and ANGEL is a time consuming process, since each tool has to be developed from the ground up. This means that considerable development time, testing time and manpower has been consumed by these projects. However, with the advent of sophisticated CBR shells, it has become unnecessary to develop a CBR application from the ground up. CBR shells are able to provide the basic components of a CBR system to the CBR application developer. These CBR shells also have mature user interfaces and are highly customizable by design. The positive implications for the CBR developer are significant and allow the CBR application developer to concentrate on his/her particular CBR application without having to develop an entire CBR environment. A discussion on CBR tools follows in the next section.

4.4 A Review of CBR Shells, Tools and an Overview of CBR Application Considerations.

A CBR shell can be defined as customizable tool for developing a CBR application. The advent of CBR shells has meant that CBR applications can be developed more rapidly. CBR shells have expedited the development of CBR applications by removing the responsibility of creating the interface and logic systems from the developer. The more sophisticated CBR shells are highly flexible in defining and adjusting the core system logic. This would include important aspects of a CBR system such as retrieval, similarity and adaptation rules. Much progress has been made on the end-user interface of CBR applications. For example, CBR Works 4 has customizable HTML generating capability, which allows the user of the CBR application to perform online queries. The ability to interface with case data is an important consideration for any CBR system. This becomes more important as the size of the case base to be used increases. Many CBR shells provide import capabilities from such commonly used database, spreadsheet and (more recently) web-based formats like XML (eXtensible Markup Language) databases. This ability to define structured cases from raw data formats means that data can easily be acquired by the CBR system.

Watson (1996) presents a comprehensive discussion on the variety of software tools and applications in his analysis of CBR tools. This was the first attempt at a comprehensive review on this subject. It is however not extensive which creates the need to discuss the tools in further detail. In his review, Watson sets out to define what a theoretically ideal CBR tool should support. He states that a tool should not only support all the main processes of CBR, namely: Retrieval, Reuse, Revision and Retention, but also support the developers in delivering an efficient tool that is capable of interfacing with other systems.

Watson (1996) suggests the following core function set that a CBR tool should contain:

- Representation - A full range of data types must be supported, e.g. numeric, string, Boolean and symbol. The cases should be able to be constructed in a way

that is reflective of the application domain, e.g. some domains may require the use of ordered symbol hierarchies and may also benefit from object-oriented inheritance.

- Retention - The case structure should be organized in a manageable structure that facilitates efficient search and retrieval. As mentioned in Chapter 2, a balance needs to be struck between retaining the semantic richness of the case to be stored whilst at the same time maintaining efficient levels of access and retrieval of these cases.

- Retrieval - Indexing must be supported by the CBR tool, whether manual or automatic, to facilitate efficient retrieval of the cases. Watson specifies that if inductive techniques are used, then the tree that is generate should be open to inspection and alteration. Alternately, if nearest neighbor retrieval is implemented then case features should be able to be weighted and similarity measures customized if necessary.

- Revision – The CBR tool should provide a facility for adaptation of cases through some form of procedural programming language or through the use of Knowledge based techniques.

The above features pertain to the core competencies of the CBR tool. In addition to these the tool should provide the developer with functionalities that make using the tool more efficient and that provide with some flexibility in the creation and maintenance of the case base. Watson (1996) lists these as follows:

- The use of large case-bases should be supported. Retrieval times should at worst increase linearly with the number of cases.

- There should be a variety of retrieval mechanisms, which should be allowed to be used in parallel with each other if necessary.

- There should be a variety of metrics available to assist in the development of the most efficient system possible and for the maintenance of the case base.

- A CBR tool should be able to provide database connectivity to a full range database formats.

- A CBR tool should provide a good user interface for the developer and the end-user alike.

- Embedded application support should be provided to assist in the embedding of the developed application. The tool should thus provide support for function libraries (e.g. DLL support) or support for the use of standard communication protocols such as DDE (Dynamic Data Exchange).

Both developer centric factors and CBR competency factors as discussed above will be used as the basis to the review of selected CBR tools, which follows in this chapter. Two of the most prominent CBR tools available for CBR application development were selected for review purposes. The selection was based on availability of the product and financial constraints since a licensed version of ESTEEM was accessible as was an evaluation version of CBR Works 4. An independent comparative review was necessary as the analysis conducted by Watson (1996) is not current enough to have considered CBR Works 4 and not detailed enough for tool selection purposes.

The CBR shells that were selected as possible development environments for developing a software cost estimation tool were CBR Works 4 and ESTEEM 1.4. Their relative strengths, weaknesses and features will be analyzed and a conclusion drawn for each along the lines of Watson's (1996) criteria.

4.4.1 ESTEEM

ESTEEM is a commercial CBR development tool that provides an environment that is not only suitable for experienced CBR developers but also for those with little programming experience. The latest version of the ESTEEM product is version 1.4 released in 1995. ESTEEM was originally built on the foundation of Intellicorp's Kappa-PC and has since been ported to C++ (Watson and Marir, 1994). ESTEEM has been particularly popular in the academic CBR research environment for CBR application development.

ESTEEM has a GUI (Graphical User Interface) based user interface and has the capability to import and export cases from dBase IV, Lotus 1-2-3 or ASCII format files. From this perspective it can be considered slightly limited in not providing a standardized database access format (The importance of this will be discussed further with respect to CBR-Works 4).

The GUI end user interface to Esteem 1.4 is divided into three main operational areas:

- Target Case Entry Screen – here target case data is entered
- Retrieved Case Screen – here retrieved case are displayed and can be analyzed I more detail in the selected case screen
- Selected Case Screen

ESTEEM can be said to lack modern features and user interface niceties, however the decision-making capabilities it has are extensive and powerful. A listing of the internal logic features that make ESTEEM so popular follows:

- ESTEEM has its own inference engine, which enables CBR application developers to create adaptation rules;
- It supports case hierarchies that helps to systematically reduce the size (and therefore complexity) of search field;

- It has the ability to access multiple case-bases and nested cases. The implications of being able to reference nested cases is that an entire tree of cases can be referenced with a single attribute entry in the case base;
- It is capable of hierarchical retrieval, allowing the induction process to be controlled through feature counting, weighted feature computation and inferred computation.

In addition to the core capabilities of ESTEEM, there are further end-user interface features that are useful to the CBR developer. Like any CBR shell should provide, ESTEEM has an end-user interface whereby the user can enter a new case, retrieve similar cases from a case-base, modify or adapt selected cases for use in the new problem situation. There exists a limited capability for the CBR developer to develop customized GUI (Graphical User Interface) interfaces for the end-user. However, the developer of an ESTEEM CBR application is able to select the capabilities that the end-user can have in the application and what modifications can be made to the case base. The user is able to adjust the way cases are retrieved to suit his/her requirements, add new cases to the case-base, use rules of the domain to assist in adaptation of a retrieved case to the new case (ESTEEM 1.4 Product Documentation, 1995).

ESTEEM has the following predefined similarity matching capabilities for cases:

- Feature counting
- Weighted feature computation
- Inferred feature computation

The above three methods are then accentuated by combinations of matching performed at the feature (attribute) level:

- Exact feature value match (and derivatives)
- Partial feature value match (and derivatives)
- Partial word matching (and derivatives)
- Equal feature matching (describes numerical equality)

- Range feature matching (similarity between numbers with a given tolerance, value returned is 0 or 1)
- Fuzzy Range feature matching
- Inferred Feature matching
- Recursive feature matching (especially for nested cases)
- Matching for multi-valued sets in text features

With reference to the weighted feature computation, ESTEEM version 1.4 has what is called an automatic weight generation facility, whereby two different heuristics are provided to determine the relative weights of a case base's features: The ID3 Weight Generation method (Quinlan, 1990) and the Gradient Descent Weight Generation method.

As the name suggests, the ID3 method employs the popular (in algorithmics) algorithm to build a decision tree for the cases in the case base. This tree is then used to calculate weights for the features that were used in the tree. Note that the user is able to select the target and source features. With the ID3 algorithm, only one target feature can be specified. Unlike the ID3 algorithm, the Gradient Descent Weight Generation method, multiple targets can be specified. The Gradient Descent Weight Generation method's algorithm is as follows: several random cases are selected from the case base, and the cases that are most similar to them (based on the current weights of the source features) are found.

Esteem makes the source Kappa-PC code available to developers. This allows the developer to embed CBR functionality within the Kappa environment. The Kappa-Pc code can be exported as C code then compiled into a standalone application (Watson, 1996).

The Capabilities of ESTEEM Contextualized by Watson's (1996) Criteria:

According to Watson's (1996) criteria for the core competencies of a CBR tool, ESTEEM's features can be summarized as follows:

- Representation - An extensive range of data types are supported which is sufficient for any CBR application.
- Retention - Like most CBR tools, cases are represented with a set of feature types and feature value pairs. These can be actively displayed and navigated through the GUI
- Retrieval – ESTEEM makes available a number of algorithms for retrieval. Combining the available forms of matching and similarity leads to greater precision. ESTEEM has provided an extensive range of similarity options within a rigid framework, in that it would have been preferable to have allowed the developer to be able to develop his/her own similarity measures in addition tot hose already provided.
- Revision – ESTEEM allows developers to create their own adaptation rules

With reference to the developer centric aids that ESTEEM provides:

ESTEEM is able to support large databases. It does however have a restrictive support for only a few database types e.g. Lotus and DBASE. This means that the developer might have to transfer the database to a compatible database format. The GUI is functional yet dated owing to its similarity to that of the Windows 3.1 operating system.

Retrieval algorithms provided (as mentioned above) are extensive and sufficient for most CBR applications. Metrics for efficiency are not provided.

The ESTEEM based application can be embedded and can exist as a standalone application.

In summary, Esteem 1.4 is a competently designed CBR shell. The areas in which it is not ideal are:

- In the flexibility of its similarity definitions
- Its limited capability for interfacing with external databases.
- Dated GUI

4.4.2 CBR Works

CBR Works was developed by Tec-Inno Germany. It makes use of an object-oriented framework for the definition and storage of cases. It does this through the use of CQL (Case Query Language), which is a standardized exchange format specially developed for CBR applications. CQL is an object-oriented language for storing and exchanging domain model descriptions and cases.

The latest release (version 4.2.1) was out in the year 2000. According to the CBR Works 4 reference manual, the most significant features of CBR Works 4 are as follows:

- Availability for major operating systems (Windows, UNIX, Macintosh)
- Object oriented modeling capabilities
- Direct ODBC (Open DataBase Connectivity) to databases
- Generic web-interface
- Flexible similarity measures
- Integration with existing systems via CQL
- HMTL and Java front end capability

CBR-Works 4 has a GUI interface that has been divided into four self-contained units:

- The concept manager – for managing and defining the structure of the cases
- The type manager – defines types for attributes and their similarity measures
- The case explorer – provides tools to manage the case base
- The case navigator – allows retrieval to be carried out

These intuitive units make it easier for the CBR developer to construct, navigate and modify the CBR application. The concept manager allows for a case structure to be imported or developed from an existing database that is ODBC compliant. Furthermore, the case explorer allows cases to be directly imported from any ODBC database. Queries are easily performed through the case navigator. All queries are totally customizable and a form of automated question answering query can be designed and performed.

In CBR Works, concepts define the structure of cases. Each concept consists of attributes, which in turn may be concepts themselves. The structure of concept attributes includes three properties in addition to the attributes definition. These are relative weight (to other concept attributes), whether an attribute is a discriminant for retrieval and whether the attribute is mandatory for queries.

Like ESTEEM, CBR Works has similarity matching capabilities for cases (concepts) and attributes. Similarity for concepts is approached from two different perspectives. This depends on whether the similarity is to be based on content or structure; contents-based similarity or structure-based similarity. Contents-based similarity is computed based on the attributes that are defined in the case structure (concept) and in CBR Works can be of four types:

- Average - where all attribute similarities are averaged.
- Euclidean - where a geometric interpretation of the distance between the contents of a concept is made.
- Minimum - where the lowest attribute similarity defines the contents-based similarity.
- Maximum – where the highest attribute similarity defines the contents-based similarity.

Concept similarity can be computed as a weighted sum of contents based similarity and structure-based similarity. In structure based similarity the similarity is computed

independent of the concepts contents. Types or attributes are defined hierarchically with CBR Works and new attributes are defined as sub-types to the elementary predefined types. Similarity can computed for these attributes in a variety of ways:

- Value to value comparisons
- Type dependant similarity – such as strings
- Functional specification – highly flexible method achieved through the use of graphs which can be tailored to the specific situation. For example, the similarity can vary as the value of the attribute increases
- User defined – through the use of the programming function.

CBR Works has the capability of defining more than one similarity for each type. This specifically caters for situations where the decision on what similarity to use

CBR Works takes a slightly different approach to similarity than ESTEEM. Even though value and type similarity is computed in the same way, CBR Works provides additional functionality. In CBR shells, the definition of similarity is generally closely linked to the definition of attribute/object types. In CBR Works, there are three initial options for similarity: standard, advanced and programming. Standard and advanced modes provide a choice of symmetric and asymmetric similarity. In symmetric mode, the result of the similarity is independent of the roles of the target and source queries, whereas in asymmetric similarity, the result depends on the role of the compared values. What makes CBR Works extremely flexible is that it has a programming mode (mentioned above), which allows the developer to write his/her own similarity functions in the Smalltalk programming environment. Furthermore the ability to compute customizable non-linear similarities is a powerful feature.

The Capabilities of CBR Works 4 Contextualized by Watson's (1996) Criteria:

As with ESTEEM and most flexible CBR tools intended for diverse CBR application development, the core functionalities of retrieve, retention, reuse and revision are well implemented:

- Representation - One again extensive range of data types are supported, which is sufficient for any CBR application. Furthermore, CBR Works is able to import case types from a database or the user is able to define his/her own type, which is not present.
- Retention - Like ESTEEM, cases are represented with a set of feature types and feature value pairs. These can be actively displayed and navigated through the GUI which displays them in a tree-like fashion. The structure can be edited and maintained through the intuitive interface.
- Retrieval - CBR Works makes available all the popular algorithms for retrieval. The developer is able to compose similarity functions manually if necessary. This is not ideal but the feature is most welcomed.
- Revision - CBR Works provides developers with capability of defining customized adaptation rules to facilitate case revision.

With reference to the developer centric aids that CBR Works 4 provides:

Like ESTEEM, CBR Works is able to support large databases. Furthermore CBR Works has Microsoft ODBC support which allows it to interface with any ODBC compliant database. This is a particularly useful feature for importing large case bases. The CBR Works GUI is intuitive and relatively easy to master.

Retrieval algorithms provided are highly customizable and the ability to generate ones own retrieval criteria enhances CBR Works appeal in terms of flexibility. As with ESTEEM, metrics for efficiency are not provided. CBR Works is intended as a complete CBR package and does not allow for standalone applications to be developed. It does

however have the capability to act as a case server through the use of CQL (Case Query Language). This means that an online web accessible application can be developed.

In summary, CBR Works provides numerous advantageous over its competitors. The most noteworthy are that similarity can be manually defined or programmed if necessary. The ODBC database support allows for the easy definition of the case structure and importing of the case base. In addition the intuitive GUI makes the development and maintenance of the case base a more efficient task. It is for these reasons that CBR Works was selected as the CBR tool to develop a CBR software cost estimation tool.

4.5 Concluding Remarks on the Application of Case-Based Reasoning to Software Cost Estimation

The purpose of the previous sections were to discuss the current state of research into analogy-based software cost estimation models and to describe CBR tools that facilitate the development of an application of that nature. Researchers such as Shepperd and Schofield (1997) and Mukhopadhyay et al (1992) have demonstrated the applicability of analogy-based models to the problem of software cost estimation. Furthermore, Briand et al (2000) and Finnie and Wittig (1997) have demonstrated CBR tools can successfully be applied to developing analogy-based software cost estimation models. The research, literature and theory discussed in this chapter and preceding chapters gives credence to and forms a foundation for the development of an experimental CBR software cost estimation model. The development of this CBR system is presented in Chapter 5.

Chapter 5: Experimental Design and Results

In this chapter the goals, definition, design and formulation of the experiments are explored in detail. In order to justify the experiment, it is necessary to place the experiment in context with similar research that has been performed by other researchers.

5.1 Placing the Experiment in Context with Similar Research through Further Exploration of the Published Findings of Other Researchers

Chapter 4 inter alia explored the various analogy-based applications developed for software cost estimation. Further to this, the underlying methodologies of the investigations into CBR as an effort estimation method shall be discussed below, through further exploration of the published findings of other researchers. Particular attention is given to the structure of the case bases used, the adaptation methods and measures of predictive performance used. Most often, detailed information on adaptation rules used and specific case structures was not available in the published literature. The research projects discussed below have all attempted to validate the use of CBR as an effort estimation technique. Usually, the CBR method was compared to some regression model or across multiple datasets.

As mentioned earlier, Jefferey and Walkerdon (1999) investigated the use of analogy and regression for estimating software effort. Their prototype ACE, estimated the effort required for a target project by searching through a database of metrics for completed projects, determining the highest ranking completed project and adjusting the effort value of the completed project taking into account the difference in size between the target and completed project. The results were then compared to a linear regression model and the results from the ANGEL (Shepperd and Schofield, 1995) estimation tool. Conclusions were then drawn from the statistical ranking of the accuracy of the various methods used. The measurement of accuracy in this experiment was MARE (Mean Average Relative Error). In addition to this they measured the prediction accuracy through the use of the

prediction at level l (Pred(l)) measurement. The use of this measure of predictive accuracy is discussed in the experimental methodology section below.

Kododa et al (2000) explored some of the practical issues associated with the use of CBR for effort estimation. They wanted to determine the impact of case base size on prediction accuracy, what number of analogies to use in making a prediction and what adaptation strategy to use in finalizing the prediction. They made use of the ANGEL estimation tool (Shepperd and Schofield, 1995) on a dataset which was divided into 4 portions of increasing size. The larger portions were a superset of the smaller portions. This was necessary to explore the impact of the size of the case base on prediction accuracy. One of the datasets supplied by Desharnais consisted of 77 software projects. Kododa et al (2000) made use of the MMRE (Mean Magnitude of Relative Error) statistic to measure the accuracy of the CBR system.

Finnie and Wittig (1997), in their investigation of software cost estimation techniques, used a project dataset of approximately 300 completed software projects to compare the accuracy of the models. They removed 50 cases from the case base to test the accuracy of the CBR system on the remaining cases. The CBR system was developed using the CBR tool Remind version 1.3. Since the data used consisted of function point counts, the case base was structured accordingly and consisted of: significant general characteristics (GSC's), system size (in unadjusted function points) and the ratios of inputs, outputs, inquiries, internal files and external files to the unadjusted function point count. The adaptation rule developed for the system was based on the differences in productivity for each significant GSC. Productivity was measured as a ratio of UFP to total development hours. Six significant GSC's were selected based on statistical analysis (refer to Chapter 3 for a full description of the GSC's used in function point analysis or to Table 5.2). A simple linear regression model was developed for each significant GSC against the productivity. The adaptation rules adjusted the effort estimation using a multiplier based on the differences for any significant GSC. After the adaptation rules were implemented, the final effort prediction was based on the average of the three nearest cases (Finnie and Wittig, 1997). Accuracy of the models developed for this experiment were measured by

the MARE (Mean Absolute Residual Error) statistic. They based the choice of this method of accuracy measure on similar research conducted by Srinivasan and Fisher (1994). After analysis of the formulation of the MARE accuracy measure, it was found that it is of the same mathematical form as the MMRE accuracy measure and therefore simply another nomenclature for the same accuracy measure.

Schofield and Kitchenham (1996) made use of the ANGEL estimation tool on a number of datasets to demonstrate the superiority of CBR in this field against regression models. They partitioned the datasets used, limiting the case base to those projects developed using the same development environment. Accuracy was measured by the MMRE statistic. Similar work was carried out by Shepperd and Schofield (1997). The final effort estimate was derived using a weighted or unweighted average of up to three analogous projects.

In all of the abovementioned research, the case similarity mechanism operated by minimizing Euclidean distance. This is considered the best approach for this problem domain (Shepperd and Schofield, 1997). A number of observations can be made from the above discussion on the methodologies used in previous research conducted into the use of CBR as an effort estimation technique. The following methodological considerations can be drawn from the literature:

- Identification of a suitable dataset containing data on past software development projects, their characteristics and the effort associated with them.
- The selection or development of an appropriate CBR or analogical tool.
- Deciding on the measure of quality for the results obtained from the tool used.
- Comparing the results with another estimation model (usually a regression model).

None of the above sources reports directly on an investigation into the role of case granularity on the quality of the results. This provided the motivation for one of the

major sub-goals of this research. The goals of the experiment are discussed in the following section

5.2 Goals of the Experiment and Hypothesis

The primary objective of this experiment was to investigate the applicability of case-based reasoning to the problem of software effort/cost estimation.

Sub-goals of the Experiment

The primary sub-goal of this experiment was to investigate the impact of case granularity on the predictive capabilities of the CBR model.

Secondary sub-goals were:

- To identify suitable case structures for the CBR system
- To select an appropriate measure of accuracy of the CBR system.

This research concludes with recommendations on the applicability of CBR to effort estimation and suggestions for future research. The logical process followed in developing the experiment is indicated in Figure 5.1

Formulation of Hypothesis for the Primary Sub-Goal

This hypothesis is based on assumption that providing the CBR system with more data regarding the software projects will improve estimation accuracy. Finnie and Wittig (1997) found that only six GSC's in their data were relevant to their effort estimate. This was determined through a statistical analysis of the 14 GSC's contained in the data. They sought to identify key factors affecting productivity and thus the most significant factors contributing to the effort estimate. In their experiment, Wittig and Finnie (1997) chose to exclude more specific data on the function point counts of External Inputs, External Outputs, External Inquiries, Internal Logical Files and External Interface Files. The use

of the full adjusted function point count was excluded from any rules and calculations used for the effort estimate. Finnie and Wittig's statistical refining of the data therefore shielded the CBR system from having to differentiate the raw data.

It is therefore necessary to determine whether providing the CBR system with a more detailed description of the software projects (cases) will necessarily result in improved prediction accuracy or whether it will reduce this accuracy as the CBR system as it becomes unable to differentiate the cases adequately for the purposes of retrieval and ultimately effort estimation.

Thus, the hypothesis can formerly be stated as follows: *Increased case granularity will improve the quality of the predictions made.*

5.3 Methodology of the Experiment

To accomplish the above stated goals a case-based reasoning model was developed using the CBR Works version 4 development environment. A number of case structures and associated case bases were constructed so as to reflect increasing granularity of the data made available to the CBR system for reasoning. Statistical analysis of the results included the evaluation of Mean Magnitude of Relative Error (MMRE) and the predictive capabilities of the model. A discussion of the methods used to obtain these statistics follows in Section 5.3.2.

The development structure of the experiment is illustrated in a flow chart depicted in Figure 5.1 below. An important aspect of any CBR system is the case structure. The case structure determines what view the CBR system has over the problem domain. The form and nature of the case structure itself is dependant on the data that is available from which to formulate it. Thus, an examination of the data was undertaken, to establish which attributes in the data should be included in the case structure. Since the hypothesis required a number of case structures to reflect the increasing granularity of the data, three case structures were formulated from the data. The following section describes the nature

of the dataset used, adjustments made to the dataset in preparation for the experiment, the measures of accuracy used, the case structures developed, the adaptation rules used and the similarity measure used for in retrieving the closest matching case.

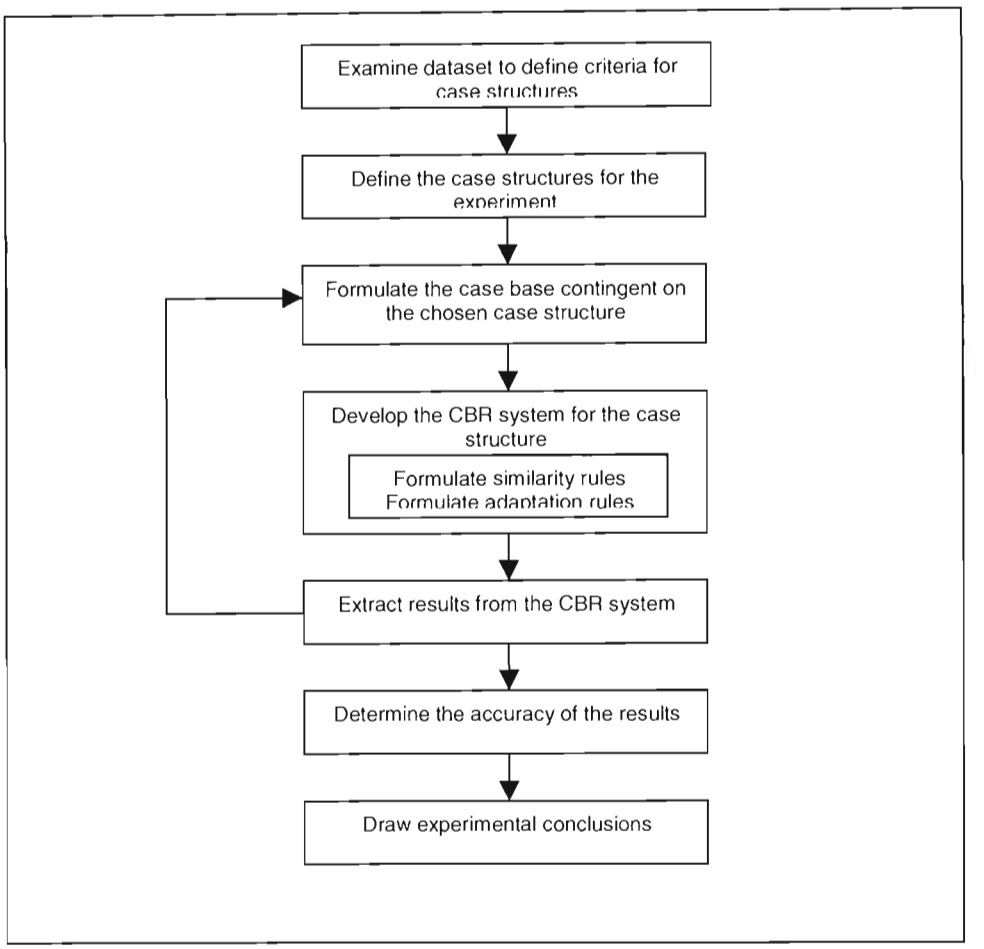

Figure 5.1 Flow chart of logical process followed in developing the experimental CBR systems.

5.3.1. Dataset Considerations

For the purposes of this research, a database of approximately 300 software projects was obtained from a Canadian researcher, Jean-Marc Desharnais (author of the database). This dataset was obtained through personal correspondence with Desharnais. A smaller subset of the Desharnais dataset has previously been made available to other researchers

in CBR. For example, Shepperd and Schofield (1995,1997) have used this data extensively in their research into the use of analogical reasoning using their ANGEL estimation tool. In a highly competitive industry, software developers are not eager to provide even academic researchers with information regarding their software projects. The datasets that are available to researchers are extremely difficult to obtain, are all of limited size (typically less than 100 projects) and are usually based on very old (early 1980's) project data. Some of these datasets have been mentioned in the discussion on software cost estimation techniques and are summarized in Table 5.1. Dolado (1999) complained of the simplistic nature of the data available to researchers, which causes any results obtained to be scientifically inconclusive. These inadequate datasets have been used in most of the research into CBR and analogical reasoning applied to software effort estimation.

Table 5.1 Datasets that have been used by researchers

Name of Dataset	Number of projects	Description	Year
Albrecht	24	IBM DP Services projects	1983
Atkinson	21	Builds to a large telecoms product	1994
Desharnais 1	77	Commercial projects from Canadian software house	1988
Finnish	38	IS projects from 9 different Finnish companies	Unknown
Kemerer	15	Large business applications	1987
Mermaid	28	New enhancements and projects	Unknown
Desharnais 2	300	Compilation of commercial projects	1997
ESA	166	European Space Agency projects	2000

It is evident from the above table that most of the datasets that have been used in the literature are based on information from a limited number of projects and are mostly contemporarily out of date.

The newer Desharnais dataset (referred to as Desharnais 2) consists of function point data for software development projects collected over a period from 1984 to 1990. It contains detailed data on the function point counts for these projects including general system characteristics (GSC's) and the total realized development effort for each project. Table 5.2 shows the structure of the Desharnais 2 dataset used. Note that the function counts are proceeded by a description and the multiplicative weighting recommended by the

International Function Point Users Group (IFPUG) for the calculation of the unadjusted function point count.

Analysis of the Dataset

After an analysis of the dataset and the fields contained in it, a number of fields were excluded from use in this experiment (these are indicated in bold in Table 5.2) for the following reasons:

- The data on project manager experience and technical personnel experience was not consistently available for every software project, therefore including this feature as a discriminant in the CBR system could compromise consistency.
- Information on exactly which sector of activities the project data was collected from was not available for privacy reasons. Desharnais indicated in which sector the projects were deployed by allocating a different number to each sector. In order to produce an accurate measure of the software productivity associated with each sector, further information on the nature of the software development would have to be known. This attribute was therefore excluded from the experimental dataset.
- The data on the number of personnel in the software development projects described in the Desharnais database was not considered for inclusion in the experimental CBR systems, since the values provided were approximate team sizes and therefore the precise effects on productivity could not be determined. An additional consideration for this decision was that the effect on development team size on programmer productivity is a subject of differing opinion in the literature (Boehm, 1981; Conte et al, 1986, Wittig 1991).
- Project duration in months and the year in which the project was completed were not considered since they of an informative nature and their inclusion would not enhance the predictive capabilities of the estimation model.

The adjustment factor based on language was considered an important contributor to software productivity. It was therefore incorporated in the calculation of the value adjustment factor (VAF) in a case structure to facilitate a more accurate adaptation strategy (see Case Structure 3). Adjustments made to the dataset included: the calculation of the weighted totals for each of the five function point counts (External Outputs, External Inputs, Internal Logical files, External Inquiries and External Interface Files). Since the data provided was of the raw (unweighted) function point counts, it was necessary to apply the IFPUG weighting scheme to these counts. The weights exist to equalize the contribution of the function counts to the UFP count. This enables the CBR system to consider each function as contributing equally to productivity and indirectly the effort total. This is an important consideration for determining the similarity of the cases.

The weighted function point counts were used to calculate the unadjusted function point (UFP) count for each project. Chapter 3 presents a detailed description of function point counting. The general system characteristics and the adjustment factor for programming language was totaled and used to calculate the value adjustment factor as per the IFPUG defined method. The IFPUG methodology only accounts for 14 adjustment factors. The addition of another adjustment factor (Adjustment factor for programming language) meant that it was necessary to adjust the IFPUG method that calculates the adjusted function point count.

With reference to Table 5.2, the definition of the function point data contained within the dataset is covered in detail in Chapter 3, Section 3.2.4.1. This section also details how the function point counts are derived.

Table 5.2 Original structure of the Desharnais dataset.

Desharnais Dataset Structure	Description of Dataset Fields	Weight/Range
Project Number	A unique number for each project	
Year Completed	Year in which the project was completed	
Duration in months	The Duration of the project in months	
Effort in Hours	Total effort in hours to complete the project	
Internal Logical Files Low Count	User Identifiable group of logically related data or control information within the boundary of the application	7
Internal Logical Files Average Count		10
Internal Logical Files Low Count		15
External Interface Files Low Count	User Identifiable group of logically related data or control information referenced by the application	5
External Interface Files Average Count		7
External Interface Files High Count		10
External Outputs Low Count	An elementary process that processes data or control information that comes from outside the application boundary	4
External Outputs Average Count		5
External Outputs High Count		7
External Inputs Low Count	An elementary process that processes data or control information that comes from within the application boundary	3
External Inputs Average Count		4
External Inputs High Count		6
External Inquiries Low Count	An elementary process that sends data or control information outside the application boundary	3
External Inquiries Average Count		4
External Inquiries High Count		6
General System Characteristic 1	Complexity of Data Communications	0-5
General System Characteristic 2	Complexity of Distributed Data Processing	0-5
General System Characteristic 3	Complexity of Performance Requirements	0-5
General System Characteristic 4	Degree of computer resource restrictions	0-5
General System Characteristic 5	Degree of Transaction Rates required	0-5
General System Characteristic 6	Degree of Online Data Entry	0-5
General System Characteristic 7	Degree of End-User Efficiency	0-5
General System Characteristic 8	Degree of internal logic files updating	0-5
General System Characteristic 9	Degree Complex Processing required	0-5
General System Characteristic 10	Degree of Reusability required	0-5
General System Characteristic 11	Degree of Installation Ease	0-5
General System Characteristic 12	Degree of Operational Ease	0-5
General System Characteristic 13	Degree of requirements for Multiple Sites	0-5
General System Characteristic 14	Degree of modifiability required	0-5
Adjustment Factor based on language	Complexity of programming language used	0-5
Total for all Adjustment Factors	Value Adjustment Factor (VAF)	
Programming Language	Category of programming language used	1-3
Project Manager Experience in Years	Experience of the senior project manager	
Technical Personnel Experience in Years	Experience of senior technical personnel	
Sector of Activity	Sector category for application deployment	1-4
Number of People Involved in the Project	Total number of people used in the project	

5.3.2. Definition of the Precision of the Experimental Results

The selection of which accuracy measures to use was based upon the analysis of quality measurements used in other similar research as indicated in the beginning of this chapter.

In comparing the performance of ESTOR with COCOMO and Function Point Analysis (FPA), Mukhopadhyay et al (1992) chose to determine some measure of accuracy and consistency. These were based on the same criteria that Kemerer (1987) used in his analysis of cost estimation models. Accuracy is said to be the most important consideration, since the software project is directly affected by the effort estimation (Mukhopadhyay et al, 1992). Mair et al (1999) support this view and state that the potential spread of error generated by an estimation system is of most significance to developers. It is necessary to compensate for the absolute error for project size that can arise. For this reason a percentage-based accuracy measure would be preferable. For example, a larger software project may have as much as a 1000-hour prediction error whereas a smaller development may only produce an error of 100 hours. The relative errors are equally important to the software developers.

In the CBR research domain, two most popular methods of establishing a models accuracy are through the calculation of MRE (Magnitude of Relative Error) and Pred(l) (Prediction at level l) (Conte et al, 1986). Both are percentage-based measures since it is necessary to compensate for the absolute error for project size that can arise - this is particularly necessary for cost estimation since project size can vary significantly. MRE was first suggested by Thebaut (1983) and has since been supported in the literature as an appropriate accuracy measurement (Conte et al, 1986; Kemerer, 1987; Mukhopadhyay 1992). However this measure has come under some criticism for weighting the importance of overestimates more than underestimates Shepperd and Schofield, 1997).

The Magnitude of Relative Error (MRE) is a normalized measure of the discrepancy between actual values V_A and fitted (predicted) values V_F (MacDonnell and Gray, 1996). This can be formulated as follows in Equation 5.1:

$$MRE = \frac{|V_A - V_F|}{V_A} \tag{5.1}$$

Mean Magnitude of Relative Error (MMRE) is the percentage-based measure of the above equation that considers all observations in the validation sample i.e. an average of the MRE results for the sample (MacDonnell and Gray, 1996). A lower MMRE will therefore indicate that the model is more accurate.

Prediction at level l (Pred(l)) is a threshold oriented (and therefore configurable) statistical measure based on the MRE measurement for each estimate. It is able to show the consistency of the predictions made by a software estimation model. It is formulated in Equation 5.2:

$$\Pr ed(l) = \frac{e}{n} \tag{5.2}$$

where l is the selected threshold value, e is the number of data points with MRE less than or equal to l. n is the total number of data points available. Conte et al (1986) state that Pred(25) is an appropriate calibration for Equation 5.2, since it is desirable that the model to be tested produces a high percentage of predictions that fall within the 25% MMRE range.

Shepperd and Schofield (1997) summarize the two techniques as follows: MMRE is a fairly conservative measure which exhibits bias against overestimates while Pred(l) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate. It is for the reasons of comparison that they chose to adopt MMRE and Pred(25) as their accuracy measurements. Indeed, Mair et al (1999), Finnie and Wittig (1997), Mukhopadhyay et al (1992) are some researchers that have used these measurements in their estimation systems. Thus, it is for the reasons of conformity and

comparability that these measurements shall be used to benchmark the CBR system developed in the course of this research.

In addition to the two techniques of measuring the quality of results, a linear regression model was developed to further validate the results obtained from the experiment.

5.4. Case Structures, Similarity Rules and Adaptation Mechanisms Implemented in the Experiment.

Three case structures were proposed to measure the accuracy and predictive capability of the CBR system and the effect that case granularity has on the quality of the predictions made by the CBR system. For each case structure the relationship between project development effort and the attributes used, is indicated in diagrammatic form. In addition, a flat file view of each case structure is given.

Three of the most important aspects considered in the literature when developing a CBR system form the basis for the discussion on the three CBR systems developed. These are:

- Definition of case structure
- Definition of similarity mechanism
- Definition of adaptation mechanism

Each case structure required the development of a new CBR system with a new case base reflecting the new case structure. In addition, similarity and adaptation rules were re-defined for each of the three CBR systems developed. The adaptation mechanism used in each CBR system is discussed in the description of the case structures. The similarity algorithm used was the same for all three CBR systems to allow for comparability.

5.4.1 Similarity Algorithm Used

In all case structures described below only the relevant function point counts were used as discriminants by the CBR system in determining the closest matching cases. The similarity measure described below was chosen because it has similarly been successfully applied by other researchers (Briand et al, 2000; Wieczonek, 2001; Finnie and Wittig, 1997 and Shepperd and Schofield, 1995, 1997). The similarity measure used to determine the closest matching cases is implemented by the following two-stage algorithm:

- Determine attribute similarity - determine the similarity of each discriminant attribute to the same attribute from the query (input) case using the following algorithm (CBR Works reference manual, 2000):

Let q = query-value (an attribute input value from the new problem case)
Let c = value of a case attribute
Let lb = the defined lower bound
Let ub = the defined upper bound

If $q = c$ then similarity = *1,0* //they are identical
else
(

$$similarity = 1{,}0 - \left(\frac{|q - c|}{(ub - lb)} \right) \tag{5.3}$$

)

The upper and lower bounds were defined to be the maximum and minimum values (respectively) of the attributes over the entire dataset.

Alternately this can be defined mathematically as in Briand et al (2000):

For each variable k of n variables included in the similarity measure, the distance $(\delta(P_{ik}, P_{jk}))$ between the target project (new case) variable value P_{ik} and the source project variable value P_{jk} is defined as:

$$\delta(P_{ik}, P_{jk}) = \begin{cases} \left(\frac{|P_{ik} - P_{jk}|}{\max_k - \min_k}\right)^2, if\ k\ is\ continuous \\ 0, if\ k\ is\ categorical\ AND\ P_{ik} = P_{jk} \\ 1, if\ k\ is\ categorical\ AND\ P_{ik} \neq P_{jk} \end{cases} \quad (5.4)$$

Where max and min are the defined upper and lower bounds of the variable respectively.

- Determine the closest matching case - which case exhibits the closest similarity to the query case.

Here the similarity of the cases is computed based on the geometric interpretation of the attribute-based similarity. Normalized un-weighted Euclidean distance is used to determine the proximity of the cases in n-dimensional space, where each dimension corresponds to a different case attribute i.e. minimization of Euclidean distance determines the closest matching case. Following from Equation 5.5, mathematically, the Euclidean distance in n-dimensions between two points P_i and P_j is (Briand et al, 2000):

$$distance(P_i, P_j) = \sqrt{\frac{\sum_{k=1}^{n} \delta(P_{ik} - P_{jk})^2}{n}} \quad (5.5)$$

Where p_i (or q_i) is the coordinate of p (or q) in dimension i.

Shepperd and Schofield (1997) summarize the benefits of the Euclidean distance method of determining case similarity as follows: Determining the proximity of cases in n-dimensional case is intuitively appealing for finding case similarity. Since each

dimension is standardized (between 0 and 1), each attribute has the same degree of influence and therefore the method is immune to choice of units and individual attribute-specific size differences. Moreover, the notion of distance gives a good indication of the degree of similarity.

The case structures derived for the purposes of this experiment are outlined in detail in the discussion on the CBR systems developed that follows. The case structure and adaptation mechanism are the defining features of the experimental CBR systems developed. The similarity measures implemented were the same for each CBR system in accordance with the above discussion on similarity.

5.4.2 CBR system 1 – Case Structure and Adaptation Mechanism

This case structure was developed to provide the CBR system with a limited view of the relation between the function point counts for data functions and transactional functions to total development effort in hours. Only five function point attributes were used for this case structure: External Interface Files (EIF) total, Internal Logical Files (ILF) total, External Inputs (EI) total, External Outputs (EO) total and External Inquiries (EQ) total.

As previously mentioned and detailed in Chapter 3; Section 3.2.4.1, these totals are derived from the low, average and high raw counts for each function multiplied by their associated weight as per the International Function Point Users Group (IFPUG) method. Included in the case structure are the unadjusted function point (UFP) count for each case and the development time in hours for each case. This is used in the final calculation for predicted effort. The relation between total effort and these function point totals is given in Figure 5.2. A flat file view of the case structure is given in the Table 5.3

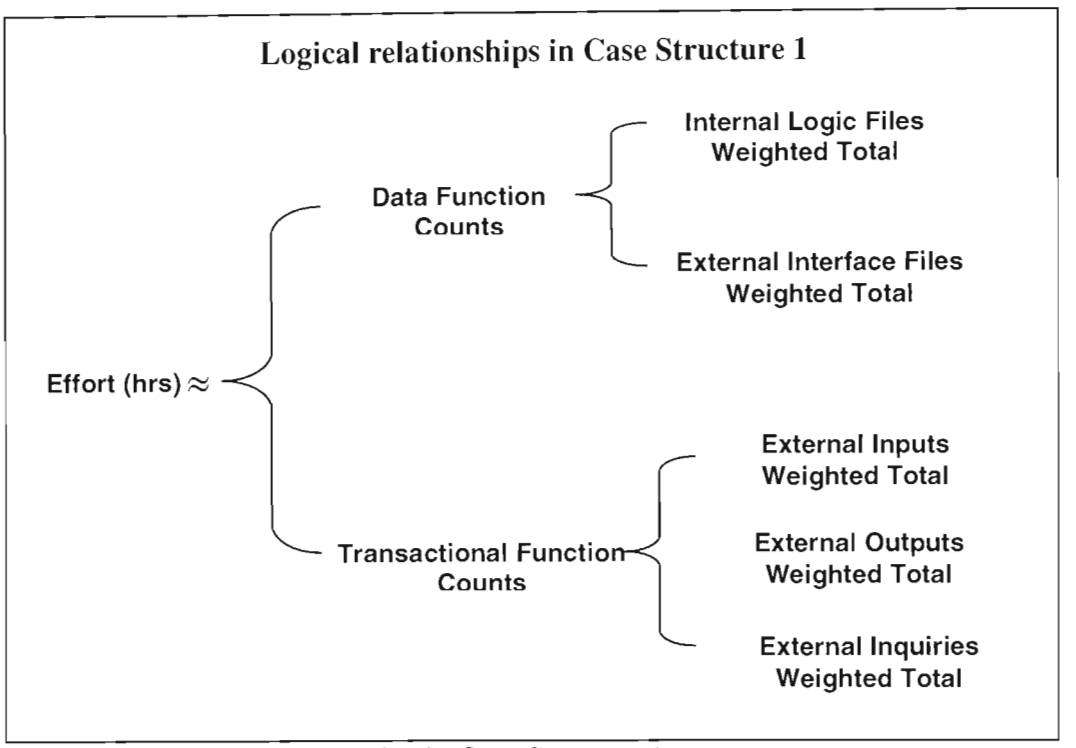

Figure 5.2 Logical relationships in Case Structure 1

Table 5.3 Flat file view of case structure 1

Case Structure 1
Internal Logical Files Weighted Total
External Interface Files Weighted Total
External Outputs Weighted Total
External Outputs Weighted Total
External Inquiries Weighted Total
Total Effort (hrs)
Total Unadjusted Function Points

The ANGEL software tool developed by Schofield and Shepperd (1995) does not implicity implement an adaptation mechanism. However, more recent research pays attention this issue (Jeffery and Walkerden, 1999; Briand et al, 2000). It was therefore decided to implement the adaptation mechanism described below.

Adaptation Mechanism

The predicted effort was calculated via linear extrapolation along the dimension of the unadjusted function point count, since effort can be strongly correlated to function points (Jeffrey and Walkerden, 1999;Briand et al, 2000). This is implemented in Equation 5.6:

$$Effort_{TARGET} = \frac{Effort_{ANALOGIUE}}{UFP_{ANALOGUE}} \times UFP_{TARGET} \quad (5.6)$$

Where, $Effort_{TARGET}$ is the predicted effort in hours for the input case, $Effort_{ANALOGUE}$ is the effort in hours of the retrieved case, $UFP_{ANALOGUE}$ is the unadjusted function point count of the retrieved case and UFP_{TARGET} is the unadjusted function point count of the input case. This is calculated by taking a ratio of the retrieved (analogue) cases effort to its UFP count, which is then re-adjusted for the target case (input case).

Case structure 1 was developed to provide the lowest system granularity. In order to increase the granularity to satisfy the requirements of the hypothesis, case structure 2 was developed, providing the CBR system with a more detailed view of the specifics of each function count.

5.4.3 CBR system 2 – Case Structure and Adaptation Mechanism

Case structure 2 was designed to increase the granularity of the data used for the effort estimation. The addition of a more detailed description of each function point count should theoretically provide the CBR system with additional information with which to make a more informed decision.

The individual scores for each function point were included in the structure. This means that for each of the five function point counts, (Internal Logical files, External Interface Files, External Inputs, External Outputs and External Inquiries) there are attributes for the individual low, average and high counts. Similarly to case structure 1, there are attributes for total development effort (in hours) and UFP counts for each case. The same adaptation mechanism was used as in case structure 1. Therefore, the predicted effort is calculated in the same manner as in Equation 5.3 above. The relation between total effort

and these weighted function point count is given in Figure 5.1. A flat file view of the case structure is given in the Table 5.3

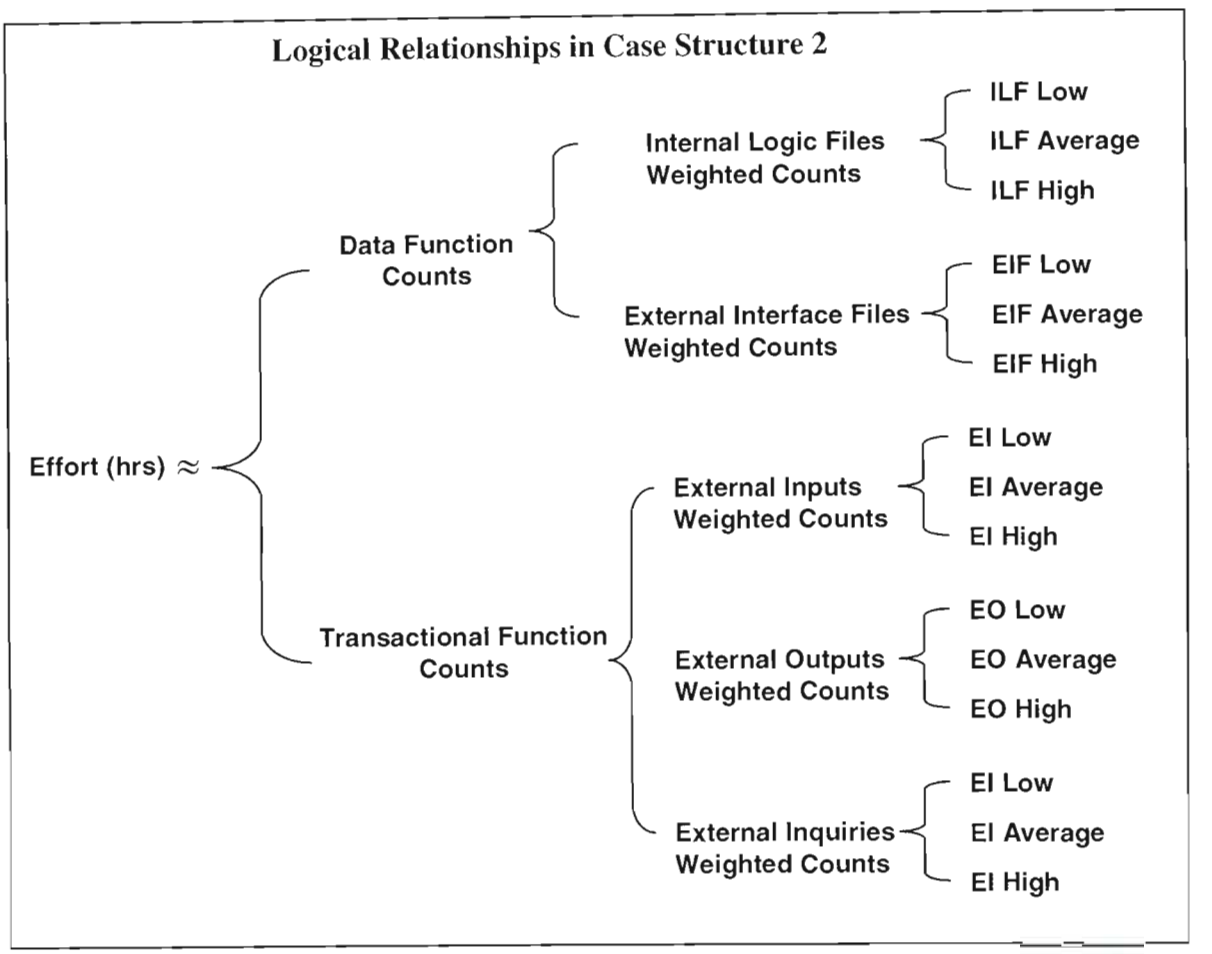

Figure 5.3 Logical relationships in Case Structure 2

Case structure 3 in CBR system 3, is an advancement on the above CBR systems (1 and 2) since it implicitly takes into account of the effect of the Value Adjustment Factor (VAF) on calculated effort. A discussion on how the inclusion of the VAF allows for an improvement in adaptation mechanism follows in the discussion of CBR system 3.

Table 5.4 Flat file view of case structure 2

Case Structure 2
Internal Logical Files Weighted Low Count
Internal Logical Files Weighted Average Count
Internal Logical Files Weighted Low Count
External Interface Files Weighted Low Count
External Interface Files Weighted Average Count
External Interface Files Weighted High Count
External Outputs Weighted Low Count
External Outputs Weighted Average Count
External Outputs Weighted High Count
External Inputs Weighted Low Count
External Inputs Weighted Average Count
External Inputs Weighted High Count
External Inquiries Weighted Low Count
External Inquiries Weighted Average Count
External Inquiries Weighted High Count
Total Effort (hrs)
Total Unadjusted Function Points

5.4.4 CBR system 3 – Case Structure and Adaptation Mechanism

Case structure 3, includes all the attributes of case structure 2, with the addition of the Value Adjustment Factor (VAF) for each case. The VAF is calculated by: totaling the general system characteristics scores, these are then multiplied by a constant of 0.01 and added to a base score of 0.70 as per the IFPUG (International function Point Users Group) method. The base score would usually be 0.65, however the inclusion of the effect of programming language means that the base score increases to 0.70. The increment of 0.05 is because each general system characteristic is scored out of a total of 5. Refer to Chapter 3; Section 3.2.4.1, for a detailed description of function point methods. This VAF is used to adjust the performance of Equation 5.3 to be more representative of the actual effort required. The relationships of total effort and the function points and adjustment factor used is indicated in figure 5.3. A flat file view of case structure 3 is given in Table 5.4.

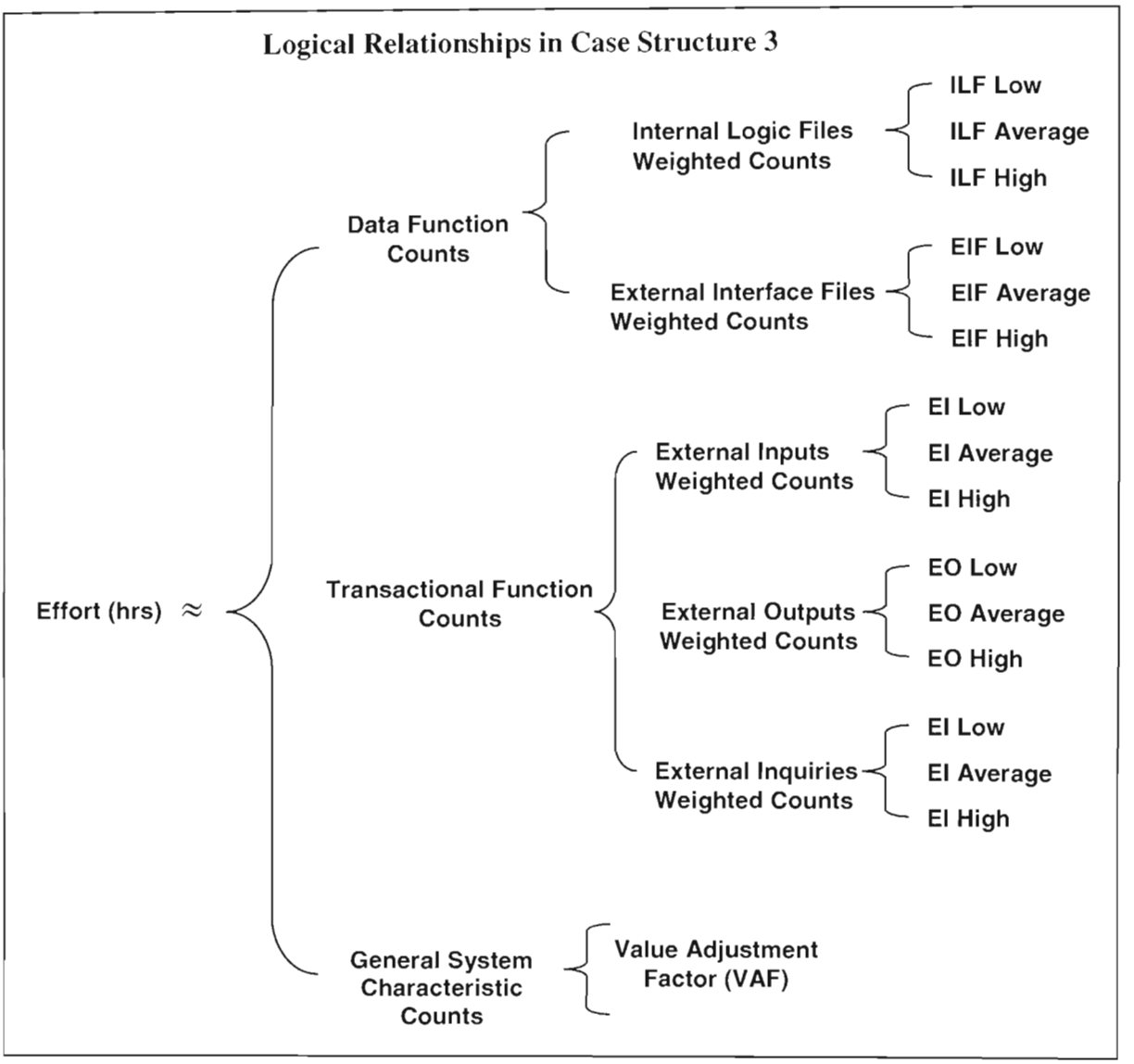

Figure 5.3 Logical relationships in Case Structure 3

Table 5.4 Flat file view of case structure 3

Case Structure 3
Internal Logical Files Weighted Low Count
Internal Logical Files Weighted Average Count
Internal Logical Files Weighted Low Count
External Interface Files Weighted Low Count
External Interface Files Weighted Average Count
External Interface Files Weighted High Count
External Outputs Weighted Low Count
External Outputs Weighted Average Count
External Outputs Weighted High Count
External Inputs Weighted Low Count
External Inputs Weighted Average Count
External Inputs Weighted High Count
External Inquiries Weighted Low Count
External Inquiries Weighted Average Count
External Inquiries Weighted High Count
Total Effort (hrs)
Total Unadjusted Function Points
Value Added Function (VAF) Adjustment Factor

Since the case structure described in Table 5.4 is different from case structures 1 and 2, an adjustment to the adaptation mechanism can be made. This is described below.

Adaptation Mechanism

This adaptation mechanism takes into account the effect of the value adjustment factor (VAF) on productivity. In the adaptation mechanism in Equation 5.7, The Unadjusted Function Point (UFP) count for each case is adjusted by multiplying it by the associated VAF for that case. This is referred to by the International Function Point Users Group (IFPUG) as the adjusted function point count. Adjusted function point counts are more reflective of development effort since factors effecting productivity are taken into account.

A modification of Equation 5.6 is used in Equation 5.7:

$$Effort_{TARGET} = \left(\frac{Effort_{ANALOGIUE}}{UFP_{ANALOGUE} \times VAF_{ANALOGUE}} \right) \times \left(UFP_{TARGET} \times VAF_{TARGET} \right) \quad (5.7)$$

Where, $Effort_{TARGET}$ is the predicted effort in hours for the input case, $Effort_{ANALOGUE}$ is the effort in hours of the retrieved case, $UFP_{ANALOGUE}$ is the unadjusted function point count of the retrieved case and UFP_{TARGET} is the unadjusted function point count of the input case. $VAF_{ANALOGUE}$ is the calculated VAF value for the retrieved case and VAF_{TARGET} is the VAF value of the problem case.

5.5 Data Collection Method

Shepperd and Schofield (1997) made use of a technique known as Jack Knifing to obtain objective results from their CBR system. They summarize the Jack Knifing technique as a validation technique whereby each case is removed from the dataset and the remainder of the cases are used to predict a solution for the removed case. The case is then returned to the dataset and the next case removed. This procedure continues until all of the selected cases have been tested. This is a popular method of testing, since the test case is actually a validated case (since it is based on actual knowledge) and the dataset is tested over itself. However, this method has traditionally been applied where the case bases have typically contained less than 80 software projects (Shepperd and Schofield, 1995, 1997).

Since the test database was based on data from 300 software development projects, using this method would have proved impractical. Finnie and Wittig (1997) have made use of the same database in their comparison of software cost estimation techniques. They selected 50 projects from the original dataset, which they used for testing purposes. This selection was excluded from the case base and used to validate the predictions made.

A similar method was adopted in this research; 50 random cases were excluded from the dataset and used as test inputs for the CBR system. The random function used was the automatic random generation function provided by Microsoft Excel version 9. For each

case the predicted versus actual effort was recorded. This information was tabularized to reflect the predicted effort in direct comparison to the actual effort for each test case. The MRE and Pred(25) was calculated for each case. The results obtained are described the following section.

5.6 Validation of Results through Comparison with Regression Models

In the literature, a variety of regressions models are frequently used to validate the accuracy of the CBR system developed. Shepperd and Schofield (1997), Briand et al (2000) and Wieczonek (2001) are some of the researchers who have made use of regression models with which to compare their CBR systems. The form of the regression technique used is usually ordinary least-squares regression. This form of regression was used to derive the models discussed below.

As in Briand et al (2000) and Wieczonek (2001), the relationship between effort and system size was modeled. The variable used for system size was Unadjusted Function Points (UFP). Wieczonek (2001) showed that an exponential relationship was the most plausible of all the possible regression models. Finnie and Wittig (1997) similarly found that an exponential regression model produced the best results. In this experiment, three regression models were developed; they were of the log-linear (exponential), logarithmic and linear forms respectively. Their derived equations are presented in Table 5.5. These models were developed from the sample data (after the 50 random cases were removed). The test data (50 random cases) was used as inputs to the regression models. Residuals (actual values) were compared directly with the predicted values to compute the MRE values and prediction accuracy with Pred(25). The co-efficient of determination, R^2, which is a measure of the percentage of variability in the data that the model can explain, is indicated for each model in Table 5.5

Table 5.5 Regression models developed with their associated equations and R^2 values.

Regression Model	Derived Equation	R^2(%)
Log Linear	$y=2787.3e^{0.0014x}$	27.60
Logarithmic	$y=7246.8\text{Ln}(x) - 33383$	27.06
Linear	$y=14.315x + 2489.6$	24.33

5.7 Results and Discussion

This section describes and discusses the results that have been obtained through using CBR to estimate software cost. These results are compared to the regression model developed and to results by other researchers in this field. Prediction accuracy (as detailed in Section 5.3.2.) is assessed using the Mean Magnitude of Relative Error (MMRE) statistic. This is complemented by the Prediction at Level (l) (Pred(l) measure which is calibrated to a value of 25%. This calibration was suggested by Conte et al (1986). Pred(25) is an indication of the percentage of prediction made within an MRE of 25%. MMRE and Pred(25) differ in that they asses slightly different characteristics of a prediction system (Shepperd and Schofield, 1997). MMRE is a termed a conservative measure that looks at the mean absolute relative error, while Pred(25) is optimistic and focuses on the best predictions made (those within 25%) (Shepperd and Schofield, 1997).

The primary objective of this experiment was to empirically demonstrate the applicability of CBR to software cost estimation. The sub-goal of this experiment was to determine the effects of case granularity on the quality of the predictions made by the CBR system. The hypothesis is restated for the purposes of the discussion contained in this section.

Hypothesis: Increased case granularity should improve the precision of the estimations made.

The results obtained from the experiment using the 50 randomly sampled test cases using three different CBR systems are summarized in Table 5.6. For reference, Appendix A

contains a detailed table of results. With reference to Table 5.6, it should be noted that a lower MMRE indicates better prediction accuracy while a higher Pred(25) indicates that a higher number of predictions within a 25% MRE are made.

Table 5.6 Comparison of MMRE and Pred(25) results obtained from the 3 CBR systems and the regression models.

Accuracy Measure	CBR System 1	CBR System 2	CBR System 3	Log Linear Regression Model	Logarithmic Regression Model	Linear Regression Model
MMRE (%)	73.22	61.44	57.08	62.11	100.40	83.85
Pred(25%)	38.00	40.00	44.00	24.00	30.00	32.00

Following Table 5.4, it is evident that the MMRE performance of the software cost estimation models that implement CBR techniques are mostly superior to the regression models. Only in the case of the least accurate (in terms of MMRE) CBR model did the best regression model (in terms of MMRE) show an advantage. The Pred(25) values obtained for each model described in Table 5.4 indicates that the CBR models were able to produce better quality results than the regression models. The observation that CBR models are superior to regression models has similarly been made by other researchers in this field. (Shepperd and Schofield, 1997; Shepperd et al, 1996; and Finnie and Wittig, 1997). Finnie and Wittig (1997) state that regression models do not perform effectively in modeling the complexities involved in software development projects. Since the dataset used in this research is the same Desharnais 2 dataset used by Finnie and Wittig (1997), their explanation for this is applicable to this research: the data used in this experiment can be termed typical of real-world software developments, with an extreme variation in productivity (between 0.01 to 0.67 UFP/hr) and a considerable range of possible causes for this productivity variation.

To place the results obtained in context with those obtained by other researchers in this field, Table 4.4 is updated to include the Pred(25) statistic where available and results from the three CBR systems developed for this experiment. The outcome is Table 5.7 below.

Table 5.7 Summary of MMRE and Pred(25) results (where available) obtained from research into analogy-based software cost estimation, augmented by the three CBR systems developed in this experiment.

Analogy-Based Model / Researcher	Year	Dataset Name	No. Of projects	MMRE %	Pred(25%)
ESTOR / Mukhopadhyay	1992	Kemerer	15	52.79	-
ANGEL / Shepperd and Schofield	1995	Albrecht	24	62.00	33
		Atkinson	21	39.00	38
		Kemerer	15	62.00	40
	1997	**Desharnais 1**	**77**	**64.00**	**36**
		Finnish	38	41.00	39
		Mermaid	28	78.00	21
ACE / Jeffery and Walkerdon	1999	In house	19	55.00	24
Finnie and Wittig	**1997**	**Desharnais 2**	**299**	**36.20**	**42**
ANGEL / Mair et al	**1999**	**Desharnais 1**	**77**	**49.00**	**-**
Briand et al and Wieczonek	2000/2001	European Space Agency	166	74.00	4
BRACE / Stamelos et al	1999	ISBSG database	100+	70.37	23.84
CBR system 1	**2002**	**Desharnais 2**	**300**	**73.22**	**38**
CBR system 2		**Desharnais 2**	**300**	**61.44**	**40**
CBR system 3		**Desharnais 2**	**300**	**57.08**	**44**

The following discussion is held within the context of the results presented in Table 5.7. Strictly speaking, a direct comparison of the three CBR systems in this experiment to other analogy-based models has no scientific foundation, since the results obtained by the various researchers are highly dependant on the underlying dataset used in their models. However, a general positioning of the accuracy of the predictions made can be gained from such an exercise. Such an analysis suggests that the three CBR systems developed fall within the range of MMRE values of those obtained by other researchers. Further to this, it can be noted that the Pred(25) value is the highest of the published Pred(25) values.

A direct comparison can however be made to those software cost estimation models that make use of the same dataset or a smaller subset thereof. In this respect all three CBR systems developed in this experiment outperform all ANGEL (Shepperd and Schofield, 1995) based models that make use of the Desharnais 1 dataset (Mair et al, 1999 and Shepperd and Schofield, 1997). This is a smaller subset of the Desharnais 2 dataset that

does not show the effect of multi-organizational data on prediction accuracy, since it only contains project data from a single organization. It would therefore be expected that the effects of multi-organization data would impact negatively on the MMRE and Pred(25) results obtained (Brian et al, 2000). The term 'outperform' is simply used to indicate that the MMRE results from the three CBR systems developed, shows that they are more accurate. Furthermore the Pred(25) measurement shows that the three CBR systems make more predictions within a 25% MRE than those made by ANGEL.

Finnie and Wittig (1997) report MMRE results of 36.2 % and a Pred(25) value of 42%. Clearly the MMRE figure obtained is superior those of the three CBR systems developed. Even the best performing CBR system (CBR system 3) yields an MMRE of 57.08%. The MMRE results obtained are superior to all of the models contained in Table 5.4. There are a number of reasons that can be attributed to this:

- Firstly, the derivation of the case structure of Finnie and Wittig's (1997) CBR system was achieved through a process of optimization; As mentioned in Section 5.1, Finnie and Wittig only made use of what they termed 'significant' general system characteristics (GSC's). They applied a statistical *t-test* to analyze which GSC's were responsible for a corresponding significant difference in productivity. Productivity was measured as the ratio of unadjusted function points to effort hours for each project. They selected six GSC's to be included in the case structure, based on this statistical analysis. This process can subjectively be referred to as a process of data smoothing, thus protecting the CBR system from inconsistencies and possible inaccuracies in the underlying dataset. The implication of this is that their CBR system is highly calibrated to the underlying dataset, which explains in part the comparatively excellent results obtained.

- Secondly, the results obtained by Finnie and Wittig (1997) are based on the selection of 50 random projects, which were used for testing purposes. Since the researcher has no control over the process of random selection, it is possible to hypothesize that the 50 random projects may have been more favorable to the

results than the 50 random projects selected in this experiment for the three CBR systems. However, no scientific credence can be given to this hypothesis without a direct comparative test using similar samples and a comparable CBR system to that developed by Finnie and Witiig (1997).

- Thirdly, the adaptation rules applied by Finnie and Wittig (1997) differ in those used by the CBR systems presented in this research. Finnie and Wittig's (1997) adaptation rules were based on the calculated differences in productivity for each significant GSC. A multiplier based on the differences in score for any significant GSC adjusted the predicted effort by a multiplier based on the differences in score for any significant GSC. Further to this, in an attempt to smooth out the effect of single incorrect estimates, an average of the estimates based on the three nearest neighbors was used in the final effort estimate. This method of adaptation is more sophisticated than the linear extrapolation mechanism used by and developed for the three CBR systems in this experiment, which adjusts the predicted effort for the nearest neighbor case according to the unadjusted function point count (for CBR systems 1 and 2) and adjusted function point count (in CBR systems 3).

The Pred(25) value of 44% achieved by CBR system 3 is higher than the Pred(25) value of 42% obtained by Finnie and Wittig (1997). This is not a significant difference, as when this value is placed in the context of the 50 random project samples used in this experiment; it is evident that after decomposition of this percentage value, CBR system 3 predicted 1 project (within an MRE of 25%) more than Finnie and Wittig (1997). Significant conclusions cannot be drawn from this difference since the use of a random sample could explain this variation.

The above discussion has validated the CBR systems developed in this experiment in comparison to other similar CBR systems. In addition it has been shown that these experimental CBR systems all outperform an algorithmic model in the form of a least-squares linear regression model. A sub-goal of this experimental research was to determine the effect of case granularity on the quality of the results obtained by the CBR

systems developed in this research. It was hypothesized that an increasing case granularity would improve the quality of the predictions made. Following the discussion on the derivation of the CBR systems and their respective case structures, the experimental CBR systems developed can be ranked according to increasing granularity as follows: CBR System 1, CBR System 2 and CBR System 3. Section 5.3.2 identified the two accepted means of determining system quality through the accuracy of predictions (Conte et al, 1986); MMRE and Pred(25).

With reference to Table 5.3, it can be seen that for each instance of increased case granularity contained in the case structures of CBR systems 1 to 3, there is a corresponding increase in Pred(25) values and a corresponding decrease in MMRE values. Quantitatively, across the CBR systems developed, there is a total 16.14% decrease in MMRE and a 6% improvement in Pred(25). Since there is a difference evident in the values obtained from these measures of accuracy, it can be deduced that case granularity does affect the quality of the predictions made by the CBR systems. Furthermore, since the reductions of MMRE and increase in Pred(25) values corresponds to the increase in case granularity of the respective CBR systems, it can be concluded that increased case granularity has a positive effect on the quality of the predictions made by the CBR systems. These measurements tend to disprove the null hypothesis that case granularity has no effect on the quality of the predictions made.

As in Finnie and Wittig (1997) a paired *t*-test was performed to verify if the results obtained for the CBR systems were statistically significantly different or whether the results could be as a result of chance factors. The null hypothesis is that the differences in MRE's produced by the CBR systems are equal to zero (H_0: $\mu = 0$). Since the calculated *t*-statistic value was well below the critical *t*-value, the null hypothesis H_0 could not be rejected. This indicates that even though there are apparent significant differences in the effort estimates produced by the three CBR systems, these individual estimates are not statistically significantly different. The complete statistical results are presented in Appendix B.

Although not directly comparable to research conducted into software cost estimation models, owing to differences in the datasets used, the results of this experiment show that they are not contradictory to the general findings of other researchers (Finnie and Wittig, 1997; Shepperd and Schofield, 1995, 1997; Jeffery and Walkerdon, 1999; Briand et al, 2000; Wieczonek, 2001; Stamelos et al, 1999 and Mair et al, 1999). In addition, the results obtained from this experiment have shown the applicability of CBR to software cost estimation by indicating that a MMRE of 57.08% is achievable using case based methods on a large multi-organizational database of software projects from a wide variety of commercial sectors. Conte et al (1986) places restrictions on what he terms a good estimation model. He states that a good estimation model should produce predictions with Pred(25) $\geq$ 75% and an MMRE $\leq$ 25%. This has realistically never been realized even by experts, Kemerer (1987) in his empirical evaluation of software cost estimation models obtains a MMRE of 167.69% for function point models and 85.48% for COCOMO based models. Similarly Mukhopdhyay et al (1992) finds that Function Point Analysis can be totally inaccurate (MMRE of 618.99%) and COCOMO, unreliable (MMRE of 102.74%). These are models traditionally relied upon by software engineers for software cost estimates. Mukopdhyay et al (1992) found that even the estimates produced by the expert they employed did not fall within Conte et al's (1986) strict criteria.

Empirical validation is however not the only consideration in assessing the applicability of CBR to software cost estimation. There are a number of intrinsic advantages that CBR has over other estimation models when applied to software cost estimation:

- Explanatory value

  Since the CBR process attempts to emulate the human thought process (Reisbeck and Schank, 1989), solutions presented by CBR models can be considered to be more easily accepted by the user of the model (Mair et al, 1999). The process of adaptation is intuitively natural since an attempt is made to place the problem case in the same context as similar cases. The inherent explanatory value that CBR models have, has

implications for the adoption of CBR in an area in which confidence in estimates is a paramount concern for the software engineer.

- Configurability

Mair et al (1999) refers to configurability as the effort required to build and adjust prediction systems. The advent of CBR tools has facilitated more rapid CBR application development (Watson and Marir, 1994). According to Kododa et al (2000), configuring a CBR system for software cost estimation is a non-trivial task that requires inter-alia the development of similarity measures, adaptation mechanisms and case structure. However, these issues have become well documented in the literature (Petrak et al, 1994; Briand et al, 2000; Jeffery and Walkerden, 1999) as the research base into CBR as a cost estimation technique has grown.

5.8 Conclusion

In this chapter, the applicability of CBR to software cost estimation has been demonstrated by producing empirical (practical) and theoretical evidence. A primary sub-goal of the experiment presented in this chapter was to explore the effect of case granularity on the quality of the predictions produced by the CBR system. It was found that increased case granularity improved the quality of the results obtained. The accuracy and quality of the results were determined by statistical methods as suggested in the literature. In comparison to other analogy-based software cost estimation models, it was found that the CBR systems developed compared favorably to the best models presented in the literature. Chapter 6 is the concluding chapter in which a summary of the achievements of this research are presented and suggestions are made for future research.

Chapter 6: Conclusion

The aim of this research was to investigate the applicability of CBR to the field of software cost estimation. To perform this investigation within a suitable background of the existing body of knowledge on CBR applied to software cost estimation, required a thorough theoretical investigation into the fields that pertain to this problem area. This was conducted through literature reviews on the current state of research into the fields of: CBR, software cost estimation, CBR applied to software cost estimation and CBR tools. These literature surveys were conducted within the following framework:

- An overview of the methodology and process of CBR formed the basis of the literature review of CBR.
- A classification for software cost estimation models, derived from the available literature was used as the basis to the discussion on prominent software cost estimation models.
- An examination of the current state of research into the field of CBR applied to software cost estimation was conducted, with specific attention given to results obtained and methodologies used in developing such a system.
- An investigation into CBR tools was conducted to ascertain the requirements for such a tool and their abilities and limitations.

The practical component of this research involved the formulation of an experiment from which empirical conclusions could be drawn as to the applicability of CBR to software cost estimation. This is discussed in further detail in the following section describing how the goals of the experiment were achieved.

6.1 How the Goals of This Research Were Implemented

A number of sub-goals were formulated to satisfy the primary goal of the research, which was to determine the applicability of CBR to software cost estimation. A secondary goal was to determine the effects of case granularity on the quality of the predictions made by

the CBR system. The nature of the primary goal was such that this research could be divided into theoretical and practical components. The theoretical component involved the establishment of the theoretical foundation of the research, while the practical component necessitated the development of an experiment.

As mentioned above, the theoretical component of this research was completed through reviews of current literature and research into software cost estimation, CBR and CBR applied to software cost estimation. This theoretical foundation was used as a basis to the formulation of the experimental component of this research. The experimental component involved the development of a number of CBR systems, not only to explore the applicability of CBR to software cost estimation but also to investigate the effect of case granularity on the quality of the results obtained. Three experimental CBR systems were developed to ascertain the applicability of CBR to software cost estimation. These three CBR systems were designed in a manner, which facilitated the exploration of the sub-goal of the research. The three CBR systems were designed to reflect improved case granularity in each instance, with the 1st CBR system having the lowest case granularity and the 3rd CBR system having the maximum granularity based on the available data.

The CBR systems developed for the purposes of the experiment required the design and implementation of similarity and adaptation mechanisms to facilitate the operation of these systems. The adaptation mechanism used in each CBR system would be contingent on the case structures developed. The case structures were developed to explore the effects of case granularity on the predictions made by the CBR system. The results obtained from the CBR systems developed for the experiment were formulated to provide a mechanism for comparison to other research conducted in this field. The predictive accuracy of the CBR systems developed was empirically evaluated defining a systematic and repeatable evaluation method. This was to ensure compliance with the requirement that any scientific research conducted should be repeatable.

The investigation into effect of case granularity on the quality of the results produced by a CBR system was empirically evaluated through the comparison of the results obtained

from the three CBR systems developed. The overall applicability of CBR to software cost estimation was determined through the analysis of all the results obtained from the CBR systems developed.

The main outcomes of this research

The final determination of the applicability of CBR to software cost estimation is based on the outcome of the experiments and the operational advantages that CBR has to offer the field of software cost estimation. It was found that the developed CBR systems produced results in line with other researchers into this field. The quality of the predictions made by the best of the three CBR systems demonstrated the applicability of CBR to this problem area and indicated that CBR has potential as an alternative to other software cost estimation techniques.

The effect of case granularity on the quality of the results from a CBR system was determined through the comparison of the results from the three CBR systems developed for experiment. The comparison of the results from the three experimental CBR systems provided a clear indication that prediction accuracy and prediction error were improved with increased case granularity. This confirms the original hypothesis that case granularity improves the quality of the predictions made by a CBR system.

It is important to note that any results obtained from a CBR system are reliant on the underlying dataset used. This is because there is no control over the quality of the data entered in the database or further information on the circumstances surrounding the development of the software projects. In comparison to other researchers, the results showed that the accuracy of CBR systems developed, fell within the range of other CBR systems developed for the same purpose and outperformed them in certain areas.

6.2 The Theoretical and Practical Contributions of this Research

The known theoretical and practical contributions of this research to the to the fields of CBR and software cost estimation are summarized as follows:

Theoretical contributions:

- An investigation was undertaken into the effect of case granularity on the prediction accuracy of CBR systems applied to software cost estimation. To the best knowledge of the author, no formal investigation into case granularity has been undertaken although it has been mentioned in the course of other research into the field of CBR applied to software cost estimation (as in Kadoda et al, 2000).

- An adaptation mechanism was proposed to improve the prediction accuracy of the CBR cost estimation system. Kolodner (1993) suggested that adaptation techniques are not a primary consideration in the development of CBR systems and that most CBR systems fulfill the role of a sophisticated decision support system. This implies that the final solution construction is usually undertaken by an expert. The adaptation mechanism developed was based upon previous research undertaken by Jeffery and Walkerden (1999) but was improved upon to take into account the effect of productivity factors on the total effort required to develop a software project.

- These results can be seen to complementary to the published accounts of applying CBR to software cost estimation. Since CBR applied to software cost estimation is a relatively new development in the software engineering field (since 1992, Mukhopadhyay et al), researchers such as Shepperd and Schofield (1997) have stressed the need for further research on more representative datasets properly reflecting the operation of the software development industry.

- The methodological applicability of CBR to software cost estimation was demonstrated through: a discussion on the process and benefits of CBR, a discussion on the current state of research into the field of CBR applied to software cost estimation and the methodological benefits that CBR has to offer this problem domain.

Practical contributions of this research project:

- The applicability of CBR to software cost estimation was demonstrated empirically. The results obtained can be considered encouraging for the software engineer. They indicate that by storing project data in a case based system and developing a CBR model for software cost estimation, useful predictions can be made.

- When compared to research into software cost estimation, CBR has been shown to out perform traditional algorithmic methods. The practical implications for the software engineer are that CBR can be considered as a viable alternative to other modeling techniques.

- Another practical contribution is that this research has demonstrated that developing a software cost estimation model using a CBR tool is a viable alternative to developing an application from first principles. The advantages of using a CBR tool for application development are that adjustments can relatively easily be made to the structure of the case-base, similarity measures and adaptation measures of the CBR system.

The methodologies used and experimental results of this investigation into CBR applied to software cost estimation technique have validated the applicability of CBR to this field. CBR is particularly well suited to this field since it is a field that has historically been reliant on the predictions made my software engineering experts (Delany and

Cunningham, 2000). CBR has a number of methodological advantages over other cost estimation techniques: CBR models offer a high degree of explanatory value as the reasoning process attempts to replicate the human thought process; and CBR is highly configurable to any environment or dataset used (Mair et al, 1999). CBR tools have furthered the advantages of CBR in this area of application, since the rapid development of CBR systems is possible using these highly configurable and adaptable tools (Watson and Marir, 1994).

6.3 Possible Directions for Future Research

In this section, several of future directions are suggested for research in the field of CBR applied to software cost estimation. The suggestions made are based on observations made throughout the course of this research:

- To investigate the impact of cross-organizational data on the quality of the predictions made by a CBR software cost estimation system.

- To investigate the effects of qualitative project data on the quality of the predictions made by a CBR system. Shepperd and Schofield (1997) observe that the results obtained by researchers have been on quantitative datasets that were never intended for analogy-based reasoning. CBR is particularly proficient in using qualitative data to make predictions (Kolodner, 1993).

- To experiment further on the effects of case granularity on the predictive performance of the CBR systems. This can be accomplished by deriving different granularities from other datasets.

- To investigate and assess current adaptation techniques and provide suggestions for alternatives.

6.4 Concluding Remarks

The justification for the use of CBR for software cost estimation can be made on the basis of the failure of existing cost estimation methods (Kemerer, 1987,1993; Symons, 1988) and the inherent capabilities of CBR in such a weak theory domain (Delany and Cunningham, 2000). The importance of accurate cost estimates for software development projects cannot be overstated as this directly influences the viability and quality of the software development (Mukhopadhyay et al, 1992). Researchers have consistently reiterated that existing solutions to software cost estimation fail to produce consistently accurate results (Jeffery and Walkerdon, 1999; Schofield, 1998). CBR can be justified as a potentially successful solution to this problem area based on its successful implementation in fields that have had difficulties in modeling the causal dependencies between the problems and the associated solutions. These are most often fields where the judgment of an expert is relied upon to provide solutions.

CBR is a relatively new field of AI and the application of CBR to software cost estimation is particularly new to the domain of software engineering. To the best knowledge of the author, no formal implementations of CBR applied to software cost estimation exist in the commercial environment. Thus, commercial acceptance of a CBR solution for software cost estimation has yet to be gained. Research that has been conducted in this field, is most often based on limited information on software projects that do not reflect accurately the intricacies of modern software developments. Thus, there is considerable scope for further research into this field. CBR software cost estimation models have not yet attained the maturity that other CBR systems applied to other problem domains have attained. In order for such CBR systems to be accepted commercially, further research needs to be conducted into this area of application.

This research has empirically and methodologically shown the applicability of CBR to software cost estimation. The quality of the results obtained through experimentation are encouraging and validate the experimental methodologies used in deriving the experimental CBR systems. The formulation, structure and methodologies used in this

experimental component of the research were clearly defined so as to ensure replicability of the experimentation undertaken. Validation of the results was ensured through comparisons with a number of regression models and to results obtained by other researchers in this field. In summary, this research has demonstrated the suitability of applying CBR to a problem domain that has historically been dominated by algorithmic modeling techniques that have not performed to the satisfaction of the software development industry.

Appendices

Appendix A

Below is a table of complete MRE, MMRE and Pred(25) values for all models developed in this research.

Random Case Number	Index	CBR System 1	CBR System 2	CBR System 3	Linear Regression	Log Linear Regression Model	Logarithimic Regression Model
		MRE (%)	MRE (%)	MRE (%)	MRE (%)	MRE (%)	MRE (%)
4	1	17.14	17.14	8.68	53.25	0.38	79.75
12	2	63.83	34.84	49.66	19.74	48.12	3.61
18	3	4.40	80.83	73.83	92.58	52.15	28.51
21	4	20.69	20.77	1.50	8.81	30.98	34.35
26	5	43.27	291.70	210.80	925.52	620.82	892.00
27	6	406.37	185.63	181.72	303.54	185.24	282.00
30	7	56.35	32.10	15.98	45.28	67.36	29.17
52	8	4.20	4.20	5.61	93.60	62.41	120.88
54	9	69.91	4.57	0.39	13.58	12.04	43.17
56	10	80.17	18.08	11.23	0.91	28.77	3.95
61	11	25.21	2.77	12.95	40.11	11.19	73.19
62	12	13.96	22.61	18.13	17.41	40.49	28.08
63	13	7.08	23.89	10.25	3.35	25.15	51.64
76	14	290.99	24.57	22.75	1.45	33.41	6.94
81	15	31.31	126.04	80.35	31.85	55.53	19.48
83	16	28.04	28.04	39.75	10.33	47.75	10.23
84	17	12.25	62.25	50.92	29.59	50.02	34.50
87	18	0.41	29.22	22.14	7.73	38.56	4.58
88	19	1.34	106.06	136.28	16.89	48.21	4.78
96	20	55.67	21.91	1.93	57.58	73.56	53.05
99	21	40.02	68.10	68.52	29.97	8.52	25.07
110	22	34.55	10.58	50.55	48.43	8.97	89.75
113	23	69.84	2.24	2.21	138.50	105.62	190.95
114	24	73.32	73.32	75.03	208.48	185.09	500.20
128	25	61.90	27.08	39.10	110.26	48.20	101.24
135	26	14.63	14.63	7.99	36.19	8.71	52.02
136	27	136.10	9.11	0.46	30.58	55.37	15.92
149	28	25.52	69.79	82.30	51.68	10.89	95.70
153	29	71.38	71.38	77.94	36.49	63.02	18.99
163	30	5.73	61.12	68.36	108.50	21.32	165.36
165	31	20.31	20.31	9.26	38.00	17.95	78.65
168	32	57.92	87.03	87.78	8.14	37.26	2.37
168	33	57.92	87.03	87.78	8.14	37.26	2.37
169	34	71.78	50.19	39.51	43.40	61.25	40.27
192	35	18.50	41.12	14.36	26.58	5.67	108.10
200	36	20.41	219.76	197.45	144.11	82.27	72.20
205	37	26.67	24.67	29.35	142.60	101.58	105.26
206	38	254.31	6.97	5.64	107.29	44.52	106.52
219	39	23.09	23.09	1.83	66.38	27.40	5.40
224	40	23.93	23.93	8.33	21.74	22.34	49.15
228	41	276.08	132.62	139.66	124.41	31.64	139.22

228	42	276.08	132.62	139.66	124.41	31.64	189.22
232	43	9.81	50.06	47.19	11.98	43.75	7.58
236	44	24.31	86.79	81.02	6.93	36.43	1.09
249	45	64.16	75.94	19.64	117.13	86.24	171.04
251	46	341.93	53.72	53.03	74.25	55.11	232.04
253	47	25.88	119.42	110.52	181.53	108.78	108.20
258	48	55.75	64.69	67.30	44.13	58.43	59.62
280	49	10.26	7.98	20.17	18.19	15.73	7.94
298	50	236.49	219.33	267.05	311.16	152.61	424.92
	MMRE (%)	**73.22**	**61.44**	**57.08**	**83.85**	**62.11**	**100.40**
	Pred(25%)	**38.00**	**40.00**	**44.00**	**32.00**	**24.00**	**30.00**

Appendix B

t-statistic results obtained from the three CBR models, t-test assuming unequal variances in each case.

	CBR Sytsem 1	*CBR System 2*
Mean	73.22249471	61.43576954
Variance	9463.474132	3891.542322
Observations	50	50
Hypothesized Mean Difference	0	
df	83	
t Stat	0.721200379	
P(T<=t) one-tail	0.23640643	
t Critical one-tail	1.663420335	
P(T<=t) two-tail	0.47281286	
t Critical two-tail	1.988960321	

	CBR System 2	*CBR System 3*
Mean	61.43576954	57.07700146
Variance	3891.542322	3724.017599
Observations	50	50
Hypothesized Mean Difference	0	
df	98	
t Stat	0.353181416	
P(T<=t) one-tail	0.362355507	
t Critical one-tail	1.660550879	
P(T<=t) two-tail	0.724711013	
t Critical two-tail	1.984467417	

	CBR System 1	CBR System 3
Mean	73.22249471	57.07700146
Variance	9463.474132	3724.017599
Observations	50	50
Hypothesized Mean Difference	0	
df	82	
t Stat	0.99415756	
P(T<=t) one-tail	0.161535406	
t Critical one-tail	1.663647708	
P(T<=t) two-tail	0.323070811	
t Critical two-tail	1.989319571	