A Study of Automated Estimating of Software Cost

Steven Shouli Wang

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.
A STUDY OF AUTOMATED ESTIMATING OF SOFTWARE COST

by

Steven Shouli Wang

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
August 1993
A STUDY OF AUTOMATED ESTIMATING OF SOFTWARE COST

Steven Shouli Wang, M.S.
Western Michigan University, 1993

The objectives of this thesis are to survey software cost estimation methods and to discuss how the methods can be applied as conceptual knowledge in a software cost estimation expert database system. The various applications of expert database systems are discussed. Consequently, a new method to approach the software cost estimation has been proposed.

This proposed method is based upon a dozen years of analyzing software cost models which have been done by some dedicated scientists in universities, research organizations, and industry. A CASE tool called IASCE has been designed to assist the project manager to estimate a proposed project cost. This tool collects accurate data from current software industry so that researchers may explore new software models and metrics. IASCE offers a mechanism for long range improvements of software cost estimation. The IASCE system is discussed and explained.
ACKNOWLEDGEMENTS

First, I thank my advisor, Dr. Dionysios Kountanis for his patience, guidance and encouragement throughout the work on my thesis. I could not have made so much progress without his guidance and help in my school years. Second, to him and other members of my committee, Dr. Fred Boals and Dr. Donna Kaminski, I extend my sincere appreciation for their supervision and support.

My special thanks goes to Dr. Donald Nelson, the department chair, for his invaluable support, providing opportunity, encouragement throughout these years without which it would have been next to impossible to be here for my higher education.

I would like to thank Phyllis and Sue, the department secretaries, for their help and kindness during my school years.

I thank Dr. Donna Kaminski and Christopher A. Oliver, who made many thoughtful comments on the issue of the communication between C language and Prolog. Most importantly I wish to thank Dr. Barry W. Boehm, author of the COCOMO model, given a special permission to use anywhere up to 15 pages of COCOMO tables and formulas in this thesis.

Finally, my warmest love and appreciation to my parents, my brother, and sister, who gave full support to me from half way around the world.

Steven Shouli Wang

ii
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A study of automated estimating of software cost

Wang, Steven Shouli, M.S.
Western Michigan University, 1993
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................................................ ii

LIST OF TABLES ............................................................................................................................................... vii

LIST OF FIGURES .......................................................................................................................................... viii

CHAPTER

I. INTRODUCTION ........................................................................................................................................ 1

Software Cost Estimating Problem in Software Engineering ........................................................................ 4

Lack of Accurate Data .................................................................................................................................. 5

No Universal Estimation Method for Software Industry ......................................................................... 6

Motivation ...................................................................................................................................................... 6

Thesis Objectives ....................................................................................................................................... 7

Thesis Organization ................................................................................................................................... 8

II. SOFTWARE COST ESTIMATION METHODOLOGY .......................................................................... 10

Introduction ................................................................................................................................................... 10

Major Software Cost Estimation Techniques ............................................................................................ 13

Categories of Cost Estimating Models ........................................................................................................ 16

Historical-Experiential Models ...................................................................................................................... 16

Statistically-Based Models .......................................................................................................................... 17

Composite Model ....................................................................................................................................... 20

A Composite Model - COCOMO .................................................................................................................. 21
Table of Contents--Continued

CHAPTER

Validation of the COCOMO Model ............................................... 29
Evaluation of the COCOMO Model ............................................... 30
Tailoring COCOMO to a Particular Installation ........................... 33
Calibrating the COCOMO Nominal Effort Equations ............... 33
Calibrating the Constant Term .................................................. 34
Calibrating the Software Development Mode ......................... 37

III. EXPERT DATABASE SYSTEM .................................................. 40
Basic EDS Architectures.......................................................... 42
The Techniques and Tools Needed to Build EDS Architecture .... 44
The AI Techniques ................................................................. 45
Database Management Technique ........................................... 46
The Logic and Logic Programming Technique ....................... 48

CEEDS: A Credit Evaluation Expert Database System .......... 52
Background Perspective ........................................................... 53
The System Architecture .......................................................... 54

IV. IASCE: AN INTELLIGENT ASSISTANT TO SOFTWARE COST ESTIMATION ......................... 58
Design Goal of the IASCE System ........................................... 61
A Mechanism for Learning ....................................................... 62

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table of Contents--Continued

CHAPTER

A Mechanism for Producing a Variety of Reports........................................... 63

A User Friendly Interface........................................................................... 63

Model Evaluation...................................................................................... 64

Tailoring a Model to a Particular Installation.......................................... 65

Coupling an Expert System into a DBMS.................................................. 66

System Architecture.................................................................................. 67

Implementation of the Initial IASCE Prototype......................................... 71

Representation of Models and Metrics....................................................... 72

Meta-Rules for Representing the Control Strategy.................................... 75

Representation of the Nominal Effort Equation........................................ 76

The Inference Engine of the IASCE System............................................ 80

A Database Scheme for the IASCE Environment..................................... 82

V. CONCLUSIONS AND FUTURE RESEARCH......................................... 89

Summary and Conclusions......................................................................... 89

Future Research......................................................................................... 91

APPENDICES

A. A Sample of Expert Database System................................................. 93

B. A Sample Interface Mechanism............................................................ 99

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table of Contents--Continued

APPENDICES

C. Sample: IM.SC................................................................. 103
D. IASCE-COCOMO: A Sample Shell Knowledge Base............. 106
E. Expert SHELL1.............................................................. 119
F. A Demo of IASCE Prototype............................................. 128

REFERENCES........................................................................... 138
# LIST OF TABLES

1. Strengths and Weaknesses of Software Cost-Estimation Methods ........... 14
2. Attributes in Farr-Zagorski Model ............................................................. 18
3. COCOMO Parameters ................................................................................. 22
4. COCOMO Software Development Modes ................................................... 23
5. COCOMO Software Cost Driver Rating .................................................. 25
7. COCOMO Module Complexity Rating Versus Type of Module ................. 27
8. Performance of COCOMO on Its Own Database .................................. 30
9. Performance of BASIC COCOMO on Its Own Database ....................... 32
10. Calibrating the COCOMO Constant Term to Project Data ..................... 36
11. Calibrating the Nominal Effort Equation to Project Data ....................... 39
12. The Parameters of Equation of Various Cost Estimating Models ......... 79
14. Global Logic View of Software Development Mode Relation ........... 84
15. Global Logical View of Model Equation Relation .............................. 85
16. Global Logical View of Metrics Relation ...................................... 85
17. Global Logical View of Cost Driver Relation ...................................... 86
LIST OF FIGURES

1. The Architecture of CES ................................................................. 56
2. The Architecture of the IASCE System .......................................... 68
CHAPTER I

INTRODUCTION

Computer users first became aware of a software crisis many years ago. Software projects were being delivered far behind schedule, quality was poor, and maintenance was expensive. And as more complex software applications were found, programmers fell further behind the demand and their results were of poorer quality. The high demand and comparatively low productivity drove software costs up. Many researchers and project managers tried to find a solution to solve this problem, and a great effort had been placed since 1960. These efforts led to a new computer field, called software engineering.

Software engineering is a relatively new discipline. It seeks to devise techniques for software development. Since the first software crisis, much progress has been made in this field. A more revolutionary approach to software development is advocated by Balzer, Cheatham, and Green [1]. They begin by observing that maintenance accounts for some 80% of total life-cycle effort and that, because maintenance is performed on the source code which has already been optimized by skilled programmers, system maintainers have difficulty in understanding and enhancing that software. They suggest an automation-based software paradigm in which maintenance is performed on the specification of the software, rather than on the code implementation as is the current custom. They
believe that this approach will eliminate the difficulties that arise from complexities in the code artificially introduced by programmers. When enhancements in product are called for, they would change the specification and then re-implement the code. To be successful, the implementation process would have to be fast, reliable, and inexpensive. They believe that automated implementation support tools either already exist or can be readily constructed for this purpose. While the authors claim that this software paradigm is capable of producing orders of magnitude improvement in productivity and quality, at this time, the development of this approach is still in its infancy.

In recent years, great efforts have been placed on the interactions between Artificial Intelligence (AI) and Software Engineering (SE). Some AI techniques have been successfully applied to automatic programming [2,3,4,5,6,7,8]. In turn, these efforts have led to the design of a new class of systems, called Computer Aid Software Engineering (CASE). One of the design goals of CASE is to develop an "expert" system to generate a design and high-level language instructions from a complete specification of a user's requirements. This is in the same spirit as developing a compiler to generate machine code from a high-level language. A typical expert system consists of a knowledge base, which contains facts and rules that capture the reasoning of human experts in a given problem domain, and an inference machine, which is capable of deriving new facts from the knowledge base. When applied to software development, we can envision a series of "knowledge-based software assistants" for each of several phases of the software
life cycle, including project management, requirements analysis, implementation, validation and testing, and maintenance. For example, one objective of an Expert Maintenance Assistant would be to help the software maintainer to modify an existing program. The human maintainer decides what changes to make and initiates them, but the Maintenance Assistant identifies all parts of the system that are affected by these changes and updates them consistently.

CASE encompasses a collection of automated tools and methods that assist software engineering in the phases of the software development life cycle. CASE tools are not a replacement for any methods and are enhancements for generating quality products. Today, many CASE tools have been used in the software industry to assist SE in the phases: (a) project initiation, (b) requirement specification, (c) structure design, (d) detailed design, (e) code and unit test, (f) integration and test, (g) acceptance test, (h) maintenance, (i) project termination, and (j) product phase-out.

For instance, a CASE Tools for Information Systems by McDonnel Douglas is called ProKit*WORKBENCH (PKWB) [6]. Version 2.0 was released in 1989. This tool is a set of advanced integrated development tools that are used during the analysis and design phase of the software development life cycle. PKWB provides graphics support for data modeling, prototyping, and the creation of data flow diagrams and structure charts. These graphic models are integrated with a data dictionary that serves as a central repository of information, which was collected during analysis and design. Output from the CASE tool satisfies project
documentation requirements and fosters communication between the developer and the end user. A more challenging research work has been done by Bimson, and Burris, called Assisting Managers in Project Definition [8]. This project is designed to support the project managers to deal with dynamic schedules and project requirements. Although great efforts have been placed in the phases of the software development life cycle [3,4,5,6,7,8], to our knowledge, there is not a CASE tool for Software Cost Estimation (SCE) available.

Software Cost Estimating Problem in Software Engineering

There are several reasons that cause difficulties in developing an automated software cost estimating tool. First, techniques of estimating costs are poorly developed. More seriously, they reflect an unvoiced assumption which is quite untrue, i.e., that all will go well. Second, estimating techniques fallaciously confuse effort with progress, hiding the assumption that men and months are interchangeable. Third, because of uncertainties of estimates, software managers often lack the courteous stubbornness required to make people wait for a good product. Fourth, scheduled progress is poorly monitored. Techniques proven and routine in other engineering disciplines are considered radical innovations in software engineering. However, the most important reason why none of SCE tool has been developed using AI technique can be categorized into two problems: lack of accurate data and there is no universal estimation method available. There is no question that the software development industry is in desperate need of better
means of estimating costs and completion times, of controlling the development process, and eliminating costly errors.

Lack of Accurate Data

A study of software cost estimation by Conte, Dunsmore, and Shen [9] surveys research efforts in this area and provides a perspective on the progress that can be expected. As once, Dunsmore, and Shen state that many project managers, team leaders, systems analysts, and programmers have developed their own intuitive techniques for dealing with the difficult world in which they operate. However, many of these techniques are adequate, but most provide far less precise estimates and control than are desired.

On the other hand, there is a small, but dedicated, cadre of scientists in universities, research organizations, and industry trying to find solutions to software development problems. Most of these researchers are basing their techniques on metrics and models. But much research is frustrated by lack of accurate data. At early stage, most projects do not collect and catalogue software "quality" data. But it's intuitively clear that the quality of the product must play an important role in the determination of the final cost of software. Some of researchers developed their own model and metrics based on the project's database which collected data from a specific software development environment.
No Universal Estimation Method for Software Industry

It is clear that most software estimation models, that we will present later, require calibration, that is, determining coefficients and constants from historical data gathered in a specific environment. Most software models cannot be transported directly from one environment to another without re-calibration. Today, there is no universal cost estimation method used in the software industry.

Motivation

The problem seems to appear clear that we can collect the accurate data across the phases of the software development life cycle according to the requirement of some existing estimating models, and refine some existing cost estimating models based on these data. Better means of estimating cost will be expected to appear in the near future. On the other hand, experienced project managers do not reason in a vacuum, to solve problems, they use a mental model of their domain. If we are to embed their expertise in an automated system, the system must also use a conceptual model of SCE as a basis for intelligent problem solving. Further, Conte, Dunsmore, and Shen evaluated and validated most estimating models [9]. The authors claim that most efficient models of software cost estimating are COCOMO[10], COPMO[9], and Jensen's model [9]. Therefore, we can store these models into an automated system as the conceptual models, and an intelligent software cost estimating tool needs such models as the
foundation for intelligent reasoning about software cost estimation.

We believe that it is possible to create a universal estimation method for the software industry. However, such a tool could take years to develop and may consume a huge amount of computer resources. In order to develop such a tool, we have to investigate the problem of interaction between Expert Systems (ES) and Software Engineering (SE). In addition, a new technique in the computer field, expert database systems, should be used for this approach. It is doubtful that expert systems for automatic software cost estimation will be successfully developed in the near future that will have a significant impact on software productivity. It is reasonable, however, to expect that some restricted forms of these may be available soon to assist project managers in software cost estimation.

Thesis Objectives

The main objective of this thesis is to discuss how to design a new method for solving software cost estimation (SCE) problems by using techniques from Artificial Intelligence (AI) and its associated fields such as Expert Database Systems (EDS) and CASE tools. The emphasis is in the design of a CASE tool which is called Intelligent Assist to Software Cost Estimation (IASCE). This thesis explores how an automatic estimating system for the software cost can be built using AI techniques. It reviews the knowledge about SCE methods, EDS architectures, techniques and tools. Consequently it explains how this knowledge can be successfully used in development of the IASCE system. It also discusses
how to represent the various SCE models and metrics by using a knowledge base, and how to store into and retrieve from the traditional databases. Based on the design goal of the IASCE system and its corresponding architecture design, a prototype has been developed to verify a part of the design goal. For example, we developed a subsystem of IASCE to provide a SCE to the unexperienced project managers by using an existing COCOMO model. In other words, we only implemented the basic function of IASCE system which estimates software cost based on the existing conceptual models, since the remaining requirements of the IASCE systems could take years to develop and may consume a large amount of computer resources. In addition, research is needed in many areas before the complete IASCE system can be built. Major areas of study related to this study include SCE, EDS, and AI. However, it is doubtless that an EDS for automatic software cost estimation will be successfully developed in the near future, if the effort in development of IASCE system can be supported by some companies and research organizations.

Thesis Organization

This thesis is organized into five chapters. Chapter II reviews existing methods of Software Cost Estimation in general and explains the COCOMO model for the cost estimating problems based on these methods. The representation of this chapter gives the readers a first impression about SCE methods so that they can understand the topic and the associated problems. Chapter III
discusses the Expert Database System, which includes EDS architecture, techniques and tools. Construction methods and tools are concerned with building the IASCE system. In Chapter IV, we discuss a new approach to Software Cost Estimation, and introduce the IASCE system, an automatic SCE environment, designed to support the suggested approach to SCE in SE life cycle. The implementation of the first prototype of IASCE system is presented in this chapter. Finally Chapter V summarizes the results of the thesis and describes future extensions of the IASCE system.
CHAPTER II

SOFTWARE COST ESTIMATION METHODOLOGY

This chapter reviews fundamental concepts and technique of software cost estimation (SCE) in order to understand the IASCE system. The information contained in this chapter is based on the book, Software Engineering Metrics and Models [9], written by S. D. Conte, H. E. Dunsmore, and V. Y. Shen, and also on the book, Software Engineering Economics [10], written by Barry W. Boehm.

The introduction gives the readers a first impression of SCE methodology, its special features and its importance to software engineering. The major software cost estimation techniques are reviewed. A software cost estimation model, COCOMO, is presented in detail. It seems appropriate to present detailed information about the COCOMO methods, which includes validation of COCOMO model, evaluation of COCOMO model, and tailoring COCOMO to a particular installation. Whereas the implementation of the first prototype of IASCE system is based on COCOMO model, the presentation of this chapter gives the readers an overview of the SCE process.

Introduction

Project management is defined as a system of procedures, practices, technologies, and know-how that provides the planning, staffing, directing, and
controlling necessary to successfully manage an engineering project. Project management is a very important activity of software engineering, nevertheless software cost estimation (SCE) plays a very important role in project management activities. Barry W. Boehm gives a precise description about SCE stated in the following quote [10]:

"Clearly, we deal with limited resources. There is never enough time or money to cover all the good feature we would like to put into our software products. And even in these days of cheap hardware and virtual memory, our more significant software products must always operate within a world of limited computer power and main memory. If you have been in the software engineering field for any length of time, I am sure you can think of a number of decision situations in which you had to determine some key software product feature as a function of some limiting critical resource.

Throughout the software life cycle, there are many decision situations involving limited resources in which software engineering economics techniques provide useful assistance. To provide a feel for the nature of these economic decision issues, an example is given below for each of the major phases in the software life cycle.

1. Feasibility Phase: How much should we invest in information system analyses (user questionnaires and interviews, current-system analysis, workload characterizations, simulations, scenarios, prototypes) in order that we converge on an appropriate definition and concept of operation for the system we plan to implement?

2. Plans and Requirements Phase: How rigorously should we specify requirements? How much should we invest in requirements validation activities (automated completeness, consistency, and traceability checks, analytic models, simulations, prototypes) before proceeding to design and develop a software system?

3. Product Design Phase: Should we organize the software to make it possible to use a complex piece of existing software which generally but not completely meets our requirements?

4. Programming Phase: Given a choice between three data storage and retrieval schemes which are primarily execution time-efficient, storage efficient, and easy to modify, respectively; which of these should we choose to implement?

5. Integration and Test Phase: How much testing and formal verification should we perform on a product before releasing it to users?"
6. **Maintenance Phase:** Given an extensive list of suggested product improvements, which ones should we implement first?

7. **Phaseout:** Given an aging, hard-to-modify software product, should we replace it with a new product, restructure it, or leave it alone (p. 12)?

The earliest efforts in estimating software cost arose from the standard industrial practice of measuring productivity for industrial workers. Productivity was usually estimated as the number of source language or machine language instructions produced over a given time period, namely one day. Initially, it was assumed that productivity was constant in computing work, but subsequent findings indicated that productivity varied greatly, depending on the current state of computing technology, capabilities of the individuals, and the complexity of the problem to be solved.

Over the past years, estimation of project size, project development time, and the cost of development has been an intuitive process. Individuals have relied on their experience and the prevailing industry norms to arrive at estimates for any given project. This method seems to work well for small and relatively straightforward systems. However, if the system is big and complicated, intuition seems to break down, and we must then turn to established quantitative management approaches to estimate these parameters.

To our knowledge, several models have been proposed for estimating software cost since 1965. Today, there are more than 30 such cost models which have been published. It is highly unlikely that any two models will estimate the same cost for a given project. We also observe that, even today, almost no model
can estimate the true cost of software with any degree of certainty. Analysis reveals that these variations are influenced by many factors and the quantity of these factors by the user of the models. Consequently, to estimate software project cost and develop guidelines for deployment of manpower for a proposed project, we need to know the factors that influence the software development process at a given facility.

For software decisions, the most critical and difficult of these inputs to provide are estimates of the cost of a proposed software project. A paper by Boehm [11] surveys the major software cost estimation techniques. Dr. Boehm has been involved in the issue of the software cost estimation since 1965, and this is the reason why his expertise can be considered as the conceptual knowledge which has been put into the knowledge base of our first prototype of the IASCE system.

Major Software Cost Estimation Techniques

Boehm summarizes the relative strengths and difficulties of the major software cost estimation methods which were used in the 1984, which in our opinion are still true today. A summary of these methods is shown in Table 1. These methods are as follows:

Algorithmic Models: These methods provide one or more algorithms which produce a software cost estimate as a function of a number of variables which are considered to be the major cost drivers.

Expert Judgement: This method involves consulting one or more experts,
perhaps with the aid of an expert-consensus mechanism such as the Delphi technique.

Analogy: These methods use reasoning by analogy with one or more completed projects to relate their actual costs to a cost estimate of a similar new

<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic model</td>
<td>Objective, repeatable, analyzable formula, efficient, good for sensitivity analysis, objectivity calibrated to experience</td>
<td>Subjective inputs, assessment of exceptional circumstances calibrated to past, not future</td>
</tr>
<tr>
<td>Expert Judgment</td>
<td>Assessment of representative interactions, exceptional circumstances</td>
<td>No better than participants biases, incomplete recall</td>
</tr>
<tr>
<td>Analogy</td>
<td>Based on representative experience</td>
<td>Representative of experience</td>
</tr>
<tr>
<td>Parkinson</td>
<td>Correlates with some experience</td>
<td>Reinforces poor practice</td>
</tr>
<tr>
<td>Price to win</td>
<td>Often gets the contract</td>
<td>Generally produces large overruns</td>
</tr>
<tr>
<td>Top-down</td>
<td>System level focus, efficient</td>
<td>Less detailed basis, less stable</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>More detailed basis, more stable, fosters individual commitment</td>
<td>May overlook system level cost, requires more effort</td>
</tr>
</tbody>
</table>

project.

    Parkinson: A Parkinson principle ("work expands to fill the available volume") is invoked to equate the cost estimate to the available resources.

    Price-to-Win: Here, the cost estimate is equated to the price believed necessary to win the job (or the schedule believed necessary to be first in the market with a new product, etc.).

    Top-Down: An overall cost estimate for the project is derived from global properties of the software product. The total cost is then split up among the various components.

    Bottom-Up: Each component of the software job is separately estimated, and the results aggregated to produce an estimate for the overall job.

    The main conclusions derived are:

    1. None of the alternatives is better than the others from all aspects.

    2. The Parkinson and price-to-win methods are unacceptable and do not produce satisfactory cost estimates.

    3. The strengths and weaknesses of the other techniques are complementary (particularly the algorithmic models versus expert judgement and top-down versus bottom-up).

    4. In practice, we should use combinations of the above techniques, compare their results, and iterate on them where they differ.
Categories of Cost Estimating Models

On the other hand, Conte, Dunsmore, and Shen analyzed existing models [9]. Authors grouped models into categories depending on the methods used in driving the models. We discuss Conte, Dunsmore, and Shen's categories on the following sections:

**Historical-Experiential Models**

Conte, Dunsmore, and Shen [9] consider that most of the cost estimation methods commonly used today fall into this category. In its query form, one or more local experts are asked to make judgements about the effort required, either for the total project or for modules into which the project has been divided. In doing so, the experts rely on their own experience with similar projects or modules, on intuition, and possibly on historically maintained information about completed projects. If more than one expert is involved, then a simple or weighted average of their estimates is taken as a "best" starting estimating point. This is, of course, a subjective procedure that is highly dependent on the competence and objectivity of the estimators. Clearly such a procedure runs the risk of overlooking some especially difficult subtasks, which may be unique to the current project. On the other hand, an expert can incorporate into his estimate unique strengths or weaknesses of the local organization that would be difficult for a general purpose estimator, which is likely to be based on average organizational
characteristics.

The expert judgement and analogy methods which Boehm summarizes can be applied at either the overall system level or at the system-component level. These are commonly referred to as top-down estimating (overall system) or bottom-up estimating (system components). Therefore, the four methods of software cost estimation, which are summarized by Boehm, are used in these models.

**Statistically-Based Models**

Conte, Dunsmore, and Shen [9] state that the models that are established based on the regression analysis are Statistically-Based Models. Regression analysis is often used to determine the relationship between programming effort and other parameters. A large number of such models, both linear and nonlinear, have been proposed for effort estimation.

**Linear Statistical Models:** If we identify EST as some unit of effort such as man-months, then a general linear model will have the form

\[ EST = c_0 + \sum_{i=1}^{n} c_i x_i \quad (2.1) \]

where \( x_i \) are software attributes or factors that are believed to affect software development effort. The \( x_i \) are called cost-driver attributes. The \( c_i \) are coefficients of \( x_i \) derived by regression analysis. There are literally hundreds of factors that may affect productivity, and hence, effort. However, many of these factors are probably insignificant and can therefore be ignored. Other factors are strongly correlated and can be combined into a single factor, thus reducing the
number of factors that need to be considered in the model. Different models vary in the factors that they include. Farr and Zagorski [12] derived some linear models for effort estimation. They give several equations depending upon a priori complexity classification. One of the regression equations takes the form:

\[ \text{EST} = -188 + 2.86x_1 + 2.3x_2 + 33x_3 - 17x_4 + 10x_5 + x_6 \]  \hspace{1cm} (2.2)

where \( \text{EST} \) is the effort in man months. The attributes \( x_i \) and their possible values are given in Table 2.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Scale or Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 ) Number of instructions</td>
<td>In thousands</td>
</tr>
<tr>
<td>( x_2 ) Number of miles traveled</td>
<td>In thousands</td>
</tr>
<tr>
<td>( x_3 ) Number of document type delivered</td>
<td>Actual number</td>
</tr>
<tr>
<td>( x_4 ) System programmer experience</td>
<td>Years</td>
</tr>
<tr>
<td>( x_5 ) Number of display consoles</td>
<td>Actual number</td>
</tr>
<tr>
<td>( x_6 ) Percentage of new instructions</td>
<td>Decimal equivalent</td>
</tr>
</tbody>
</table>

In Equation (2.2), the large negative constant term will almost surely result in negative effort estimates for small- or medium-sized projects. Furthermore, the \(-17x_i\) term will certainly lead to effort underestimates when used relatively to programmers with a lot (say 10 years) of experience. These are natural consequences.
of regression and good examples of why individual coefficients should not be interpreted independent of the other terms in the equation. Conte states that, in general, linear models have not proven to be satisfactory for effort estimation. One possible explanation for this is that effort is a highly nonlinear function of a large number of variables, and this nonlinearity cannot be adequately captured in a linear model. Furthermore, the application of a regression model to projects outside the range of data from which it was derived is ill-advised: a regression model derived from single-person projects should not be applied to team projects (and vice versa).

Nonlinear Statistical Models: Most of the nonlinear models that have been studied so far can be expressed in the form

$$EST = (a + b \cdot KDSI^c) \cdot m(X) \quad (2.3)$$

where KDSI is the estimated size of the project measured in thousands of lines of delivered source code; a, b, and c, are constants usually derived by regression analysis; and m(X) is an adjustment multiplier that depends on one or more cost-driver attributes denoted by the vector X. In some cases, a, b, and c may also be functions of one or more cost drivers. Since m(X) can also be a complicated nonlinear function of several variables, equation (2.3) is too complex to lend itself readily to standard regression analysis techniques. Instead, it is more customary to derive a base-line or nominal estimator (usually based on least squares) of the form

$$EST_{nom} = a + b \cdot KDSI^c \quad (2.4)$$
and then to adjust this nominal effort with the adjustment factor $m(X)$. Some typical nominal estimators that have reported in the literature are

\[ \overline{EST} = 5.2 * KDS1^{0.91} \quad (Walston-Felix[14] \ (2.4a)) \]
\[ \overline{EST} = 3.2 * KDS1^{1.05} \quad (Boehm-Model[7] \ (2.4c)) \]
\[ \overline{EST} = 3.0 * KDS1^{1.12} \quad (Boehm-Model2[7] \ (2.4d)) \]
\[ \overline{EST} = 2.8 * KDS1^{1.20} \quad (Boehm-Model3[7] \ (2.4e)) \]
\[ \overline{EST} = 5.288 * KDS1^{1.047} \quad (for \ KDS1=10) \quad (Doty[13] \ (2.4f)) \]

Conte, Dunsmore, and Shen state that as these models imply, the primary factor affecting software cost estimation is assumed to be the size of the project, which is normally measured as lines of code in thousands. However, as Conte, Dunsmore, and Shen state when comparing models, we must be careful that the definitions of size are compatible. The main problem is whether comment lines are included in KDSI. The Boehm and Doty models do not consider comment lines. In the Walston-Felix model in Equation (2.4a), however, KDSI does include comments that constitute up 50% of the total lines of code.

**Composite Model**

Conte, Dunsmore, and Shen [9] state that composite models incorporate a combination of analytic equations, statistical data fitting (either with linear or non-linear models), and expert judgement. Such composite model is the RCA PRICE S model [14]. PRICE S uses project size, type, and complexity as primary attributes to produce a top-down estimate of the cost of system functions for each
phase of the project. The best known of all composite models is COCOMO (COstructive COst MOdel) [10], which is our key topic in this chapter.

A Composite Model - COCOMO

Conte, Dunsmore, and Shen [9] state that COCOMO is the most complete and thoroughly documented of all models for effort estimation. This is one reason why we selected COCOMO to be as the first conceptual knowledge. COCOMO provides specific formulae for estimating the development time schedule, overall development effort, effort breakdown by phase and activity, and maintenance effort.

There are three levels of COCOMO: Basic, Intermediate, and Detailed. The development effort equations are of the form

\[ EST = a_i KDSI \cdot m(X) \]  \hspace{1cm} (2.5)

where KDSI is Delivered Source Instructions in thousands excluding comments and \( m(X) \) is a composite multiplier that depends on 15 cost driver attributes. The principal cost driver in Equation (2.5) is program size. These development modes are identified by Boehm using certain criteria and the constant \( a_i \) changes with the mode and level, whereas the constant \( b_i \) changes only with the mode. These modes and the values of \( a_i \) and \( b_i \) for the Basic and Intermediate levels are shown in Table 3.

Projects of the organic mode are characterized by being relatively small in size, requiring little innovation, having relaxed delivery requirements, and being
developed in a stable in-house environment. Projects of the embedded mode are characterized by being relatively large, needing to operate within tight constraints, and having a high degree of hardware and customer interface complexity, rigid requirements, and greater need for innovation. Projects of the semidetached mode fall somewhere in between the organic and embedded modes.

In Basic COCOMO, $m(X) = 1$ for each cost driver $X$; in Intermediate COCOMO, they are assigned various values. The 15 recommended cost driver attributes are grouped into four categories which can be found in Boehm’s book [10]. Complete definitions of each cost driver are shown in Table 5 and Table 7. After defining each driver attribute, ratings are assigned to each cost driver on a five- or six-point scale from Very Low to Very High or to Extra High, depending on the degree to which that attribute applies to a given project. A numerical

<table>
<thead>
<tr>
<th>Mode</th>
<th>Basic</th>
<th>Intermediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a_i$</td>
<td>$b_i$</td>
</tr>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
</tr>
<tr>
<td>Semidetached</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Table 4
COCOMO Software Development Modes

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Organic</td>
</tr>
<tr>
<td>Organizational understanding of product objectives</td>
<td>Through</td>
</tr>
<tr>
<td>Experience in working with related software systems</td>
<td>Extensive</td>
</tr>
<tr>
<td>Need for software conformance with pre-established requirements</td>
<td>Basic</td>
</tr>
<tr>
<td>Need for software conformance with external interface specification</td>
<td>Basic</td>
</tr>
<tr>
<td>Concurrent development of associated new hardware and operational</td>
<td>Some</td>
</tr>
<tr>
<td>procedures</td>
<td></td>
</tr>
<tr>
<td>Need for innovative data processing architectures, algorithms</td>
<td>Minimal</td>
</tr>
<tr>
<td>Premium on early completion</td>
<td>Low</td>
</tr>
<tr>
<td>Product size range</td>
<td>&lt; 50 KDSI</td>
</tr>
<tr>
<td>Examples</td>
<td>Batch data reduction,</td>
</tr>
<tr>
<td></td>
<td>Scientific models, Business models, Familiar OS, compiler, Simple inventory, production control</td>
</tr>
</tbody>
</table>

value is then assigned to each rating for each attribute which is shown in Table 6.

In order to provide a reasonably concise example of a current state of the art cost estimation model, the intermediate level of COCOMO is described below.

Boehm describes that the intermediate COCOMO estimates the cost of a proposed software product in the following way:

1. A nominal development effort is estimated as a function of the product's size in delivered source instructions in thousands (KDSI) and the project's development mode.

2. A set of effort multipliers are determined from the product's ratings on a set of 15 cost driver attributes.

3. The estimated development effort is obtained by multiplying the nominal effort estimate by all of the product's effort multipliers.

4. Additional factors can be used to determine dollar costs, development schedules, phase and activity distributions, computer costs, annual maintenance costs, and other elements from the development effort estimate.

Step 1 - Nominal Effort Estimation: This step is to determine the project's development mode. The COCOMO software development modes are Organic mode, Semidetached mode, and Embedded mode according to eight different features required by a proposed project. After determining the project's development mode, there are three different resulting scaling equations for each mode in COCOMO estimation model. They are used to determine the nominal
Table 5
COCOMO Software Cost Driver Rating

<table>
<thead>
<tr>
<th>Cost Driver</th>
<th>Very low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
<th>Extra high</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELY Effect: slight inconvenience</td>
<td>Low, easily recoverable losses</td>
<td>Moderate, recoverable losses</td>
<td>High financial loss</td>
<td>Risk to human life</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA (DB bytes)/Prog. DSI) &lt; 10</td>
<td>10 = &lt; D/P &lt; 100</td>
<td>100 = &lt; D/P &lt; 1000</td>
<td>D/P ≥ 1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPLX See Table 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Computer Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME = &lt; 50% use of available execution time</td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOR = &lt; 50% use of available execution time</td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRT Major change every 12 months; Minor: 1 month Interactive</td>
<td>Major: 6 months</td>
<td>Major: 2 months</td>
<td>Major: 2 weeks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TURN</td>
<td>Minor: 2 weeks</td>
<td>Minor: 1 week</td>
<td>Minor: 2 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Personal attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACAP 15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AEXP = &lt; 4 months experience</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td>12 years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCAP 15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VEXP = &lt; 1 month experience</td>
<td>4 months</td>
<td>1 year</td>
<td>3 years</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEXP = &lt; 1 month experience</td>
<td>4 months</td>
<td>1 year</td>
<td>3 years</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODP No use</td>
<td>Beginning use</td>
<td>Some use</td>
<td>General use</td>
<td>Routine use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOOL Basic</td>
<td>Basic mini</td>
<td>Basic</td>
<td>Strong maxi</td>
<td>Add requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>microprocessor tools</td>
<td>midi/maxi tools</td>
<td>programming tools</td>
<td>design, management,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>documentation tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCED 75% of nominal</td>
<td>85%</td>
<td>100%</td>
<td>130%</td>
<td>160%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost Driver</th>
<th>Very low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very high</th>
<th>Extra high</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELY Required software reliability</td>
<td>.75</td>
<td>.88</td>
<td>1.00</td>
<td>1.15</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>DATA Data base size</td>
<td>.94</td>
<td>1.00</td>
<td>1.08</td>
<td>1.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPLX Product complexity</td>
<td>.70</td>
<td>.85</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td>1.65</td>
</tr>
<tr>
<td><strong>Computer Attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME Execution time constraint</td>
<td>1.00</td>
<td>1.11</td>
<td>1.30</td>
<td>1.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOR Main storage constraint</td>
<td>1.00</td>
<td>1.06</td>
<td>1.21</td>
<td>1.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRT Virtual machine volatility</td>
<td>.87</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TURN Computer turnaround time</td>
<td>.87</td>
<td>1.00</td>
<td>1.07</td>
<td>1.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Personal attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACAP Analyst capability</td>
<td>1.46</td>
<td>1.19</td>
<td>1.00</td>
<td>.86</td>
<td>.71</td>
<td></td>
</tr>
<tr>
<td>AEXP Applications experience</td>
<td>1.29</td>
<td>1.13</td>
<td>1.00</td>
<td>.91</td>
<td>.82</td>
<td></td>
</tr>
<tr>
<td>PCAP Programmer capability</td>
<td>1.42</td>
<td>1.17</td>
<td>1.00</td>
<td>.86</td>
<td>.70</td>
<td></td>
</tr>
<tr>
<td>VEXP Virtual machine experience</td>
<td>1.21</td>
<td>1.10</td>
<td>1.00</td>
<td>.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEXP Programming language experience</td>
<td>1.14</td>
<td>1.07</td>
<td>1.00</td>
<td>.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project attributes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOPD Use of modern programming experience</td>
<td>1.24</td>
<td>1.10</td>
<td>1.00</td>
<td>.91</td>
<td>.82</td>
<td></td>
</tr>
<tr>
<td>TOOL Use of software tools</td>
<td>1.24</td>
<td>1.10</td>
<td>1.00</td>
<td>.91</td>
<td>.83</td>
<td></td>
</tr>
<tr>
<td>SCED Required development schedule</td>
<td>1.23</td>
<td>1.08</td>
<td>1.00</td>
<td>1.04</td>
<td>1.10</td>
<td></td>
</tr>
</tbody>
</table>

## Table 7

COCOMO Module Complexity Rating Versus Type of Module

<table>
<thead>
<tr>
<th>Rating</th>
<th>Control Operations</th>
<th>Computational Operations</th>
<th>Device-dependent Operations</th>
<th>Data Management Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very low</td>
<td>Straightline code with a few nonnested SP operators: DOs, CASEs, IFTHENELSEs</td>
<td>Evaluation of simple expressions: e.g., A = B + C * (D - E)</td>
<td>Simple read, write statements with simple formats</td>
<td>Simple arrays in main memory</td>
</tr>
<tr>
<td>Low</td>
<td>Straightforward nesting of SP operators; mostly simple predicates</td>
<td>Evaluation of mode-rate level expressions, e.g., D = SORT (B**2 - 4<em>A</em>C)</td>
<td>No cognizance needed of I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap I/O processing includes device selection, status checking and error checking</td>
<td>Single file subsetting with no data structure changes, no edits, no intermediate files</td>
</tr>
<tr>
<td>Nominal</td>
<td>Mostly simple nesting. Some intermodule control, decision tables</td>
<td>Use of standard math and statistical routines; basic matrix/vector operations</td>
<td>Operations at physical I/O level (physical storage address translations: seeks, read, etc.), optimized I/O overlap</td>
<td>Multi-file input and single file output. Simple structural changes, simple edits Special purpose subroutines activated by data stream contents. Complex data restructuring at record level</td>
</tr>
<tr>
<td>High</td>
<td>Highly nested SP operators with many compound predicates. Queue and stack control. Considerable intermodule control</td>
<td>Basic numerical analysis: multivariate interpolation, ordinary differential equations; basic truncation roundoff concerns</td>
<td>Routines for interrupt diagnosis, servicing, masking. Communications line handling</td>
<td>A generalized, parameter-driven file building, command processing, search optimization Highly coupled dynamic relational structures. Natural language data management</td>
</tr>
<tr>
<td>Very high</td>
<td>Reentrant and recursive coding. Fixed-priority interrupt handling</td>
<td>Difficult but structured N.A. nearsingular matrix equations, partial differential equations</td>
<td>Device timing-dependent coding, micro-programmed operations</td>
<td></td>
</tr>
<tr>
<td>Extra high</td>
<td>Multiple resource scheduling with dynamically changing priorities. Microcode-level control</td>
<td>Difficult and unstructured N.A.; highly accurate analysis of noisy, stochastic data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

development effort for the project in man-months as a function of the project's size in KDSI. These equations are listed below.

\[
\text{Organic} \quad \text{EST}_{\text{nom}} = 3.2 (\text{KDSI})^{1.05} \quad \text{man-month}
\]

\[
\text{Semidetached} \quad \text{EST}_{\text{nom}} = 3.0 (\text{KDSI})^{1.12} \quad \text{man-month}
\]

\[
\text{Embedded} \quad \text{EST}_{\text{nom}} = 2.8 (\text{KDSI})^{1.20} \quad \text{man-month}
\]

Step 2 - Determine Effort Multipliers: Each of the 15 cost driver attributes in COCOMO has a rating scale and a set of effort multipliers which indicate by how much the nominal effort estimate must be multiplied to account for the project to work at its rating level for the attribute.

Step 3 - Estimate Development Effort: We compute the estimated development effort for the proposed software project as the nominal development effort times the product of the effort multipliers for the 15 cost driver attributes.

Step 4 - Estimate Related Project Factors: COCOMO has additional cost estimating relationships for computing the resulting dollar cost of the project and for the breakdown of cost and effort by life-cycle phase (requirements, design, etc) and by type of project activity (programming, test planning, management, etc). Further relationships support the estimation of the project's schedule and its phase distribution, for example recommended development schedule can be obtained from the estimated development man-months via the three mode schedule equation as following.

\[
\text{Organic} \quad \text{TDEV} = 2.5 (\text{KDSI})^{0.38} \quad \text{months}
\]

\[
\text{Semidetached} \quad \text{TDEV} = 2.5 (\text{KDSI})^{0.35} \quad \text{months}
\]

\[
\text{Embedded} \quad \text{TDEV} = 2.5 (\text{KDSI})^{0.32} \quad \text{months}
\]
In the book [9], Conte, Dunsmore, and Shen given the validation and evaluation of the COCOMO Model based on the certain criteria for judging the goodness of a model. Authors analyzed the strength and weakness of the COCOMO model. The following briefly discusses the issues in the COCOMO model, which is given by Conte, Dunsmore, and Shen.

Validation of the COCOMO Model

The COCOMO equation were derived from a database of 63 projects, which were completed over one period of 15 years from 1964 to 1979 primarily at TRW System, Inc. The projects were written in several different languages, including assembly language, FORTRAN, COBOL, and PL/1. The projects vary in size from about 2,000 to about 1,000,000 lines of code (excluding comments), and they vary in type from business and scientific applications to systems, control, and supervisory types. Productivity in lines of code man-month ranged from a low of 28 to a high of 1250. Cost driver attribute ratings were assigned to each project using expert judgement based on a Delphi-type technique[10]. These ratings are subject to considerable error, especially when applied to projects completed many years ago, for which complete documentation must not have been available.

The equations derived in COCOMO were not obtained directly from a least squares regression method. Instead the author used a combination of experience, results of other cost estimation models, the subjective opinion of experienced software managers, and trial-and-error to arrive at initial model parameters based on a subset of the entire database. These initial parameter values were further refined, "tuned", and calibrated using additional projects from database [9] (p. 128).

According to Conte, Dunsmore [9], and Shen's criteria, they given a Table 8 which shows the performance of COCOMO on its own validating database. MRE means the mean magnitude of the relative error, and PRED (1) means prediction at level 1. From Conte, Dunsmore, and Shen's analysis, we know that basic COCOMO does not perform well even when applied to its own database.
Intermediate (1) does much better, while Intermediate (2) gives excellent results. In addition to the 15 cost drivers described earlier, Intermediate (2) introduces another cost driver called RVOL (Requirements Volatility). This measures the changes in problem requirements after programming has begun. This cost driver multiplier has values ranging from 0.91 to 1.62. Conte, Dunsmore, and Shen based on these measures of accuracy, they applied it to its own validating database. In conclusion, Conte, Dunsmore, and Shen [9] state that this is especially true when we consider the extreme ranges of productivity evidenced in this database.

Evaluation of the COCOMO Model

Conte, Dunsmore, and Shen [9] state that any model, regardless of how it is derived, should be tested on an independent database. There have been no published reports on the performance of COCOMO as applied to independent
database. At Conte, Dunsmore, and Shen's opinion [9], a major problem in attempting to do so is that the information needed to use COCOMO is seldom available in published data. Intermediate COCOMO requires, a collection of 15 or 16 parameters in addition to the size. Furthermore, the mode parameters has 3 choice and each of the 16 attribute parameters has a choice of 4 to 6 ratings. The large number of parameter choices provides the model with great flexibility and range, but at the same time with great volatility. Conte, Dunsmore, and Shen state that the numerical values for cost driver attributes provided in [9] are as follows:

for example, we can calculate that \( \max m(X) = 72 \) while \( m(X) = 0.088 \); if we assign to each cost driver its maximum possible value, the adjusted multiplier will be 72. Thus, if for a given project

\[
EST_{nom} = 100 \times PM
\]

the model could predict an estimated effort is more than 800 times the minimum estimated effort. The extreme range of possible estimates is both a strength and a weakness of the model. In conclusion, Conte, Dunsmore, and Shen claim that major weakness is to require the manager to supply a large amount of information in order to obtain an effort estimate. As our opinion, however, it is possible to apply AI technique to solve this shortcoming of the COCOMO method, so that COCOMO model can be applied to independent database and current project estimation.

In [13], Mohanty applied 13 different cost estimation models to a hypothetical project of size 36,000 lines of code and obtained cost estimates ranging from $362,500 to $2,766,667. If we assume that the project falls in the semidetached mode, Intermediate COCOMO gives the nominal effort

\[
E_{nom} = 3.0 \times (36)^{1.12} = 166\text{MM}
\]

and using a cost per person-month of $50,000, we obtain an estimated nominal cost of $691,781. This fits comfortably in the middle of the range obtained by Mohanty (p. 134).

In [9], Conte, Dunsmore, and Shen applied Basic COCOMO to other
databases. The performance of Basic COCOMO on its own database (Set A) and other databases are given in Table 9.

The results in Table 9 are favorable to Basic COCOMO since it produces somewhat better results than Boehm's own mode classification scheme. This motivated us to consider the COCOMO model as the first conceptual knowledge which has been stored in our database.

Table 9

<table>
<thead>
<tr>
<th>Set</th>
<th>No. Projects</th>
<th>MRE</th>
<th>PRED (.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>63</td>
<td>0.48</td>
<td>0.27</td>
</tr>
<tr>
<td>B</td>
<td>33</td>
<td>0.30</td>
<td>0.55</td>
</tr>
<tr>
<td>C</td>
<td>40</td>
<td>0.45</td>
<td>0.43</td>
</tr>
<tr>
<td>D</td>
<td>19</td>
<td>0.61</td>
<td>0.32</td>
</tr>
<tr>
<td>E</td>
<td>17</td>
<td>2.12</td>
<td>0.18</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>0.22</td>
<td>0.67</td>
</tr>
</tbody>
</table>

In [10], Boehm gives more detail information about tailoring COCOMO to a particular installation. We review is basic knowledge in order to support our new method for software cost estimation. In following we briefly discuss how to tailor COCOMO to a particular installation according to Boehm's method [11]. The information contained in following sections is based on the book, Software Engineering Economics [11], written by Barry W. Boehm.
Tailoring COCOMO to a Particular Installation


In general, the COCOMO model should be a reasonable model for most software cost-estimation situations. Often, however, a given installation can develop a specially calibrated and tailored version of COCOMO which will be more accurate and easy to use within the context of the particular installation (p. 342).

Calibrating the COCOMO Nominal Effort Equations

Boehm points out that for various reasons, the nominal effort equations for the three standard COCOMO development modes may not provide the best fit for a particular installation. Boehm [11] claims that the main reasons for this are as following:

1. An installation may consistently judge the COCOMO cost driver attribute ratings by different standards than were used in calibrating COCOMO. The most common areas where this occurs are required reliability, analyst and programmer capability, and use of modern programming practices. This situation can be accommodated by recalibrating the constant term in the nominal effort equation for the installation's development modes.

2. An installation may employ consistently different definitions of "delivered," "source instructions," "development", or "man-month" than those used in COCOMO. Here again, the simplest solution would be to recalibrates the constant terms in the COCOMO nominal effort equations.

3. An installation's usual development mode may be somewhere in between the standard COCOMO development modes, in which case a special nominal effort equation can be calibrated to the installation's experience (p. 364).

In Boehm's book [10], techniques for each of these situations are given, and illustrated by an examples. We briefly review them in the following.
Calibrating the Constant Term

The simplest and most stable way to calibrate COCOMO to an installation's experience is to establish the most appropriate development mode for the installation, and to use a least-squares approximation technique to calibrate the constant term for the development mode's nominal effort equation to the installation's project data.

To be specific, Boehm supposes that the organic mode best represents the development mode of a given installation. Then he wishes to determine the most appropriate constant, \( c \), for the organic-mode nominal effort equation in the COCOMO estimating relationship

\[
MM = c(KDSI)^{1.05}\Pi(EM)
\]

where \( \Pi(EM) \) represents the overall product of the effort multipliers resulting from a project's cost driver attribute ratings, or, more concisely, its effort adjustment factor \( \Pi_{(EM)} \).

Boehm supposes that the installation has completed a number of projects \( p_1, \ldots, p_n \), whose sizes were \( KDSI_1, \ldots, KDSI_n \), whose overall effort adjustment factors were \( \Pi_1, \ldots, \Pi_n \), and whose actual development efforts were \( MM_1, \ldots, MM_n \). Then, we wish to find the value of \( c \) in the system of linear equations

\[
\begin{align*}
MM_1 &= c(KDSI_1)^{1.05}\Pi_1 \\
MM_2 &= c(KDSI_2)^{1.05}\Pi_2 \\
&\vdots
\end{align*}
\]

\[(2-5)\]
\[ MM_n = c (KDSI_n)^{1.05} \Pi_n \]

which minimizes the sum of the squares of the residual errors

\[ s = \sum_{i=1}^{n} [c (KDSI_i)^{1.05} \Pi_i - MM_i]^2 \]  \hspace{1cm} (2-6)

or for simplicity setting

\[ (KDSI_i)^{1.05} \Pi_i = Q_i \]

thus, find the value of \( c \) that

\[ s = \sum_{i=1}^{n} [cQ_i - MM_i]^2 \]

Then, determine the optimal coefficient \( c \) by setting the derivative \( ds/dc \) equal to zero and solving for \( c \).

\[ \frac{ds}{dc} = 2 \sum_{i=1}^{n} [cQ_i - MM_i] Q_i = 0 \]

\[ \sum_{i=1}^{n} c(Q_i)^2 - MM_i Q_i = 0. \]

Thus, \[ c = \frac{\sum_{i=1}^{n} Q_i}{\sum_{i=1}^{n} Q_i^2} \]  \hspace{1cm} (2-7)

Similarly calibrate the constant terms, with exponents 1.20 or 1.12, respectively, replacing the exponent 1.05 in Eq. (2-5) and the expression for \( Q_i \).

Example: Boehm also gives an example in his book [10]. In that example, he supposes that an organic-mode installation had completed and collected data on five projects, whose sizes \( KDSI_i \), effort adjustment factors \( \Pi_i \), and development efforts \( MM_i \) were those given in Table 2-7, along with the estimates produced by the standard organic-mode estimator.
The projects actual data are generally somewhat higher than the standard COCOMO estimates, so it would have been expected the optimal coefficient $c$ for the installation to be somewhat higher than the standard coefficient $c = 3.2$.

And, in fact, we have from Table 10

$$c = \frac{\sum_{i=1}^{5} \text{MMEI}_i \times \text{QI}_i}{\sum_{i=1}^{5} \text{QI}_i^2} = \frac{11,246}{2950} = 3.81$$

Thus, the tailored effort equation for the installation is

$$MME = 3.81 \times (KDSI)^{1.05} (EM)$$

### Table 10

Calibrating the COCOMO Constant Term to Project Data

<table>
<thead>
<tr>
<th>Project Number(i)</th>
<th>KDSI_i</th>
<th>$\Pi_i$</th>
<th>$MME$</th>
<th>$MME_i$</th>
<th>$Q_i$</th>
<th>$MME_iQ_i$</th>
<th>$Q_i^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.75</td>
<td>13</td>
<td>15</td>
<td>4</td>
<td>60</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1.0</td>
<td>36</td>
<td>44</td>
<td>11</td>
<td>484</td>
<td>121</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.80</td>
<td>59</td>
<td>60</td>
<td>19</td>
<td>1140</td>
<td>361</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>1.0</td>
<td>114</td>
<td>140</td>
<td>36</td>
<td>5040</td>
<td>1296</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>0.70</td>
<td>108</td>
<td>133</td>
<td>34</td>
<td>4522</td>
<td>115</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11,246</td>
<td>2950</td>
</tr>
</tbody>
</table>

Boehm in his book [10], has also given another method to show how to calibrate the software development mode, which we briefly discuss below:

Boehm states that the least-squares technique may be used to calibrate both the coefficient term $c$ and the scale factor $b$ in the COCOMO effort equation

$$\text{MM} = c(K\text{DSI})^b \Pi(EM) \quad (2-7)$$

corresponding to the definition of a new COCOMO software development mode tailored to the practice of the particular installation.

Boehm’s first step is to rearrange Eq. (2-7), and then to linearize it by taking logarithms of both sides (base-10 is used here, but natural logarithms may be used as well).

$$c(K\text{DSI})^b = \frac{\text{MM}}{\Pi} \quad (2-8)$$

$$\log c + b \log(K\text{DSI}) = \log \left( \frac{\text{MM}}{\Pi} \right) \quad (2-9)$$

Thus, if an installation has completed a number of projects $p_1, \ldots, p_n$ with size $K\text{DSI}_1, \ldots, K\text{DSI}_n$ with overall effort adjustment factors $\Pi_1, \ldots, \Pi_n$ with actual development efforts $\text{MM}_1, \ldots, \text{MM}_n$, then we wish to solve the following system of equations in order to obtain the optimal values of $\log c$ and $b$.

$$\log c + b \log(K\text{DSI})_1 = \log \left( \frac{\text{MM}}{\Pi}_1 \right)$$
$$\log c + b \log(K\text{DSI})_2 = \log \left( \frac{\text{MM}}{\Pi}_2 \right)$$
$$\log c + b \log(K\text{DSI})_n = \log \left( \frac{\text{MM}}{\Pi}_n \right) \quad (2-9)$$

Thus, we minimize the sum of the squares of the residual errors in Eq.(2-9). It is a fairly straightforward exercise in numerical analysis to determine that the
optimal values $\log c$ of logc and $b$ of $b$ may be determined by solving the equations:

\[ a_0 \log c + a_2 b = d_0 \]
\[ a_1 \log c + a_2 b = d_1 \]  \hspace{1cm} (2-10)

where the quantities $a_0$, $a_1$, $a_2$, $d_0$, and $d_1$ are calculated from the following equations:

\[
\begin{align*}
    a_0 &= n \\
    a_1 &= \sum_{i=1}^{n} \log (KDSI)_i \\
    a_2 &= \sum_{i=1}^{n} [\log (KDSI)_i]^2 \\
    d_0 &= \sum_{i=1}^{n} [\log (MM/II) ]_i \\
    d_1 &= \sum_{i=1}^{n} \log (MM/II)_i \log (KDSI)_i \\
    \log c &= \frac{a_0 d_2 - a_1 d_0}{a_0 a_2 - a_1 a_1} \\
    b &= \frac{a_0 d_2 - a_1 d_0}{a_0 a_2 - a_1 a_1}
\end{align*}
\]

Example: Boehm [10] gives data illustrated on Table 11 which show the quantities involved in calculating $\log c$ and $b$ from the project data given in the previous example.

The resulting values of $\log c$, $c$, and $b$ are then

\[
\begin{align*}
    \log c &= \frac{(7.92)(9.25) - (6.08)(11.82)}{(5)(7.92) - (6.08)^2} = 0.53 \\
    b &= \frac{(5)(11.82) - (6.08)(9.25)}{(5)(7.92) - (6.08)^2} = 1.09
\end{align*}
\]

Thus, the new COCOMO software development mode is tailored to the practice of this particular installation would have the effort equation

\[ MM = 3.39 \times (KDSI)^{1.09} \times (EM) \]
Calibrating the Nominal Effort Equation to Project Data

<table>
<thead>
<tr>
<th>Project Number</th>
<th>KDS</th>
<th>II</th>
<th>MM</th>
<th>( \log(KDS) )</th>
<th>( \log(MM) )</th>
<th>( \log(MM/II) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.75</td>
<td>15</td>
<td>0.70</td>
<td>0.49</td>
<td>1.30</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1.0</td>
<td>44</td>
<td>1.00</td>
<td>1.00</td>
<td>1.64</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.80</td>
<td>60</td>
<td>1.30</td>
<td>1.69</td>
<td>1.88</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>1.0</td>
<td>140</td>
<td>1.48</td>
<td>2.18</td>
<td>2.15</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>0.70</td>
<td>133</td>
<td>1.60</td>
<td>2.56</td>
<td>2.28</td>
</tr>
<tr>
<td>Σ</td>
<td></td>
<td></td>
<td></td>
<td>( a_1 = )</td>
<td>( a_2 = )</td>
<td>( d_0 = )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.08</td>
<td>7.92</td>
<td>9.25</td>
</tr>
</tbody>
</table>


Boehm points out that if the sample size of project data is less than 10 projects, pick a standard COCOMO development mode and recalibrate the constant term, rather than recalibrating an overall development mode. However, if the sample size of project data is more than 10 projects, then recalibrate an overall development mode.
CHAPTER III

EXPERT DATABASE SYSTEM

One of the most rapidly growing fields of artificial intelligence is the field of expert systems (ESs). Currently, ESs exist or are being constructed for medicine, business, industry, and national defense. These systems are composed of a knowledge base of rules and facts and an inference mechanism that uses the knowledge base to respond to queries posed by users. The objective of such systems is to capture the knowledge of experts in particular domains (such as aircraft engine maintenance, financial analysis, computer configuration control, blood diseases, and intelligent information systems), and make it generally available to nonexpert users and have special small knowledge bases, they are thus limited in their applications. In recent years, as ESs have expanded and are more widely used, an increasing number of their applications, including CAD/CAM, CASE, office automation and military command and control, have a requirement for knowledge-directed processing of shared information. Those knowledge bases are more difficult to manage and their major management activity has been in the access, update and control of the rules and facts.

Fortunately, on the other hand, Database Management Systems have been developed to manage large amounts of shared information in secondary storage. A DBMS can allow multiple users (application programs) access to the same
collection of information for purpose of retrieval and/or update. Under these circumstances, the DBMS also provides a separate software facility required between the users and the database to protect the shared information. This facility provides consistency control, recovery control, concurrence control and security control. In addition, a DBMS can provide other advantages in controlling data redundancy and distribution, and making application program easier to develop and maintain.

In recent years, a great effort has been placed on the interaction between the Prolog systems and relational databases [17,18,19,20,21,22,23]. These efforts have led to the design of a new class of systems, called Expert Database Systems (EDS), which use a logic programming style for formulating queries and constraints, and database technology for providing efficient and reliable access to mass-memory data [24,25,33,34,35,36].

EDS are database management systems (DBMS) endowed with knowledge and expertise to support knowledge-based applications which access large shared databases. The architectures, tools and techniques needed to build such systems are varied, and draw upon such fields as artificial intelligence, database management and logic programming. It is precisely the union of ideas from these fields that provides the coordination for new insights and tools to build intelligent software cost estimation systems.

Expertise may reside within the system to improve performance by providing intelligent question-answering, using database semantic integrity constraints for
query optimization and combining knowledge and data-driven search techniques in efficient inference schemes. Conversely, expertise may reside outside the system in knowledge-based application that interpret vast quantities of data and make decision-impelling recommendations to users. The last approach is used to build the first prototype of IASCE system. Obviously, the goal of EDS research and development is to provide tools and techniques to make database "active" agents that can learn, and reason so that allow database systems to support artificial intelligence applications that manage and access large knowledge bases and databases. It is significant to integrate EDS with software engineering (SE) to develop CASE tools.

This chapter briefly reviews the basic architectures, tools and techniques needed to build such EDS. The information contained in this chapter is based on the papers [26,27,28,29,30,31,32], written by some outstanding EDS researchers. Finally, a sample of a technique of EDS is presented in order to understand our approach to the first prototype of the IASCE system.

Basic EDS Architectures

Expert Database Systems (EDS) represent the confluence of concepts, tools and techniques from different areas: artificial intelligence (AI), database management (DB), and logic programming (LP). The special appeal of EDS is that they extract a variety of ways in which knowledge and expertise can be incorporated into system architectures. A paper [33] by Larry Kerschberg surveys
research efforts in the EDS architectures, which is expressed in the following quote [32]:

There are several possible scenarios:
(1) an expert system loosely-coupled with a database system;
(2) a database management system (DBMS) enhanced with reasoning capabilities to perform knowledge-directed problem solving;
(3) an LP system or an AI knowledge representation system, enhanced with database access and manipulation primitives;
(4) an intelligent user interface for query specification, optimization and processing; and
(5) a tightly-coupled EDS "shell" for the specification, management and manipulation of integrated knowledge databases. All of the above architecture are meaningful, and there indeed may be many others. The particular one chosen will depend on the application requirements and the availability of tools for their implementation. The terms "loosely-coupled" and "tightly-coupled" have come to characterize two important classes of EDS.

By loose coupling we mean that both the AI system and the DBMS will maintain their own functionality and will communicate through a well-defined interface. For example, an AI system might send SQL queries to the database system, and conversely, the database system might send messages to the AI system placing data onto its blackboard. In addition the DBMS could pose questions to the AI system in much same way a user might consult an expert system. Examples of such architectures include [26,27,28,29].

Tight-coupling, on the other hand, implies that at least one system has knowledge of the inner workings of its counterpart, and that special, performance-enhancing access mechanisms are provided. Applications that require tight coupling are those in which both the data and knowledge may be updated and in which the most timely changes must be accessible. Examples of such systems are POSTGRES and the work reported in [20] (pp. 151-152).

During the past few years, we have seen these EDS architectures in terms of different types of applications. As loosely-coupled applications, the most obvious approach to data/knowledge integration is to interface a knowledge-based "shell" or an AI language (e.g., Prolog) with a DBMS. Loosely-coupled applications are typically those that view the database as a data server, with knowledge-
based processing used to interpret data obtained by issuing SQL queries to the database. The amount of expertise and knowledge required to construct the interface will depend on the nature and the amount of data being retrieved, as well as the amount of preprocessing needed to formulate the query.

This architecture is easy to be used to implement a certain application, of course, a loosely-coupled architecture was used for implementing the first prototype of IASCE systems. There are many examples for using loosely-coupled architectures during the past years. For example, the MEDCLAIM system [26] system, a loosely-coupled EDS architecture was implemented, contains about 120 claims evaluation and BC/BS policy rules and a relational database of 6000 records representing medical knowledge and BC/BS policy regulations about the medical necessity and adequacy of the treatments for 1000 well-established diseases. The ES shell was used to codify problem-solving knowledge regarding valid claims and the relationships among claim items. The relational database was used to encode highly-structured and formatted knowledge regarding typical recuperation profiles and treatment plans for the 1000 most common diseases and diagnoses. Thus, both the ES shell and relational database contain different types of knowledge about the application domain. They look like two completely independent system.

The Techniques and Tools Needed to Build EDS Architecture

The techniques and tools needed to build a EDS architecture are varied,
and draw upon such fields as artificial intelligence (AI), database management (DBMS) and logic programming (LP).

The AI Techniques

Researchers in AI have become increasingly aware of the need and advantages of EDS architectures. From the AI point of view, there is a need to have knowledge based applications access large databases. As knowledge bases become larger, they pose serious system performance problems. The pattern-matching processes involved in determine which rules are candidates for "firing" are performance bottlenecks because the rules are not indexed with respect to their component predicates. Most expert systems rely on operating system virtual memory techniques, and overall performance degrades under heavy paging requirements. Thus secondary storage access mechanisms are desirable for AI systems. They can be used to index a rule base and to provide fast access to facts stored in database files. How to manage a large knowledge base still is an open issue in the AI research area, although some results have been obtained [32,33].

Another important course of AI research that impacts EDS is work concerning the knowledge level, and the insights gained by asking modal questions regarding the knowledge base and knowledge derived through inference [34,35]. By taking a functional view of a knowledge base as a knowledge server, one can "tell" and "ask" the knowledge base what it knows. One fundamental result of some AI researchers is that AI knowledge representation systems require
complicated processing and interpretation of symbolic information and axioms associated with the "world" being modeled. The processing involved is quite complex, exceeding the capabilities of current database systems; but it may be possible to characterize different types of knowledge "engines" that are amenable to support by EDS.

**Database Management Technique**

Database Management Systems (DBMS) support the organizational concept the data is a corporate resource that must be managed, refined, protected and made available to multiple users who are authorized to use it. Thus concurrent access to large shared databases is a very important research issue of DBMS. DBMS are used in increasingly more complex environments, those in which entities are related in very complex ways, and the rules governing their behavior in response to updates need to be made explicit, rather than be hidden in application programs. These requirements indicate the DBMSs should also be enhanced with reasoning capabilities for performance reasons.

For example, there is a need to manage different types of data: text, graphics, images, computer-aided design (CAD) schemes, etc. Also, DBMS are being used to support complex environments such as software engineering environments (SEE), configuration management (CM), etc. As we know, traditional DBMS are hard-pressed to handle these new duties--those supporting more semantically meaningful "data models" and new features such as extensibility and
long transactions--are being proposed.

This new class of systems is called "object-oriented database systems" (OODB) [36,37]. The goal of OODB is to provide data models whose structures, operations and constraints can deal with complex objects as they are. This implies that the objects may be hierarchical, and the operations on them need not be decomposed into simpler operations. The modeling paradigm for these systems is based on object-oriented programming as exemplified by the Smalltalk language [38].

There are many important issues to be addressed when build OODB. They are outlined in the following:

Data abstraction and encapsulation - An object class has a set of operators similar to abstract data types. In terms of the implementation interface, there must be both a public and a private interface.

Object identity - Every object has a unique identifier which is independent of the particular property values that the object may have.

Messages - Objects communicate by sending messages to one another, and each message consists of a receiver object-identifier.

Graphics - Complex objects and their methods can best be understood and manipulated through object-oriented graphics interface.

Transaction management - In many complex applications transactions may run for long time periods, and effective method are needed to handle consistency, concurrent control, recovery, etc.
There are some non-partitionable applications in which the shell must retrieve the entire fact base in order to perform its reasoning. The cache size would have to be very large. In short, the application provide certain requirements for an EDS as follows:

1. A query language and data model that can express the "semantics" of space and time;
2. The batched reintegration of updated replicated information in a possibly distributed database;
3. Efficient processing of a large number of situation action rules (triggers) over the database on the arrival of new information;
4. The optimization and processing of recursive logic rules over the database in response to query requests.

The Logic and Logic Programming Technique

Logic and logic programming (LP) is another important technique to build an EDS. Logic provides a formal basis for both relational database and database theory. Logic may be used to extend the expressive power of query language to include recursive queries that handle the transitive closure operation found in applications such as network routing problems, and CAD/CAM. In addition, logic and logic programming provide efficient mechanisms to integrate data, meta-data, that is, data about data or schema information, domain-specific knowledge and control knowledge. Now let us look at how to implement LP technique to build
an EDS, since LP techniques are used to build the first prototype of the IASCE
system.

Logic sees a database from two points of view: (1) the model theoretic
approach; and (2) the proof theoretic approach. In the model theoretic approach
the database definition, or schema, is viewed as a time-variant definition specified
by means of data structure definitions and integrity constraints. The database
state is the collection of data instances, at any time is an interpretation of that
schema. Integrity constraints are proper axioms that the database state must
satisfy at all times. Queries on the database are expressed as well-formed formu-
lae to be translated into relational operations. Traditional DBMS such as rela-
tional database systems hold on this viewpoint.

There is no separation of the schema and data in deductive database
approach. The database is represented as a first-order theory with equality.
Facts (data instances) are represented as ground well-formed formulae (wffs) of
the first-order theory. The set of proper axioms provides the wffs for deduction
and integrity constraints. The set of theorems constitutes the implicitly derivable
information. Queries are considered theorems to be proved from the axioms.

Logic programming provides a computational language, Prolog, for the
proof theoretic approach. Prolog processes Horn Clauses which are subset of
First-Order Predicate Calculus. But, Prolog is not exact logic programming
because it has several additional features not found in logic programming:

1. Control of search through depth-first-search and backtracking;
2. Built-in predicates such as Cut and Fail which can control the inference process;

3. Performance sensitivity to the order of predicates in the knowledge base, and the order of terms within predicates.

However, these Prolog features are what we need to build our first prototype of the IASCE system. In recent years, great efforts have been placed on the interaction between the Prolog systems and relational databases [21,22,23,27,30,31,36]. There many approaches have been proposed for interfacing Prolog to relational database systems. Now let us look at two interesting proposed approaches [28]:

1. With the coupling approach, an interface is built between two currently available Prolog and database systems, which preserve their individuality; the interface provides the required procedures for bringing data (facts) from the disk database into main-memory logic programming execution environment such an running expert inference engine built using Prolog.

2. With the integration approach, a single system is developed which provides Prolog as programming language and includes database management procedures. This approach requires the development of new techniques and algorithms, specifically designed.

The coupling approach is easier to achieve, and has been in fact selected for the first prototype of the IASCE system as well as in many other prototypes [40, 41,42,43,44,45]. However, coupling must be performed with extreme care.
simulation, presented in [46], shows that the performance of the interfaces is quite critical, and that the trivial solution of mapping each Prolog query to a new database query is unacceptable for several applications. Within coupling, we further can distinguish two alternative approaches:

1. With loose-coupling, the interface is activated independently from the actual Prolog inference process. Loose coupling is either performed at compile time, or on a rule-by-rule basis, prior to the activation of that rule. Loose-coupling is also called static coupling, because coupling actions are performed independent from the actual pattern of execution of rules.

2. With tight-coupling, the interaction between Prolog and database systems is driven by the inference process. The interface is activated whenever the logic programming system needs more data (facts) from the database system in order to proceed with its inference. Tight coupling is also called dynamic coupling because coupling actions are performed in the frame of the execution of each rule.

Loose and tight coupling have major differences in complexity, selectivity, memory required and performance. With loose-coupling, less queries are executed on the database, because each predicate or rule is separately considered once for all; while with tight-coupling each rule or predicate might be considered several times. However, queries in loosely coupled systems are less selective than queries in tightly coupled systems, because variables are not instantiated (bound to constants) when queries are executed. Therefore, the amount of main memory
required for storing data retrieved by a loosely coupled system is generally higher than required by a tightly coupled system. Specially, loose-coupling approach is not acceptable for a network application because the main-memory problem is a critical issue for the network systems.

The first prototype of IASCE systems uses both loose and tight coupling. In particular, loose-coupling is used whenever the database predicates in the program are highly selective, and therefore few facts need to be retrieved for a particular predicate; in this case, a small amount of core memory is sufficient for storing the facts required by the program. Tight-coupling is used for all remaining predicates. The more detailed representation will be given on Chapter IV. A sample program represented in the following section shows how we can use loose-coupling to build an EDS.

**CEEDS: A Credit Evaluation Expert Database System**

Prolog is a very powerful language that can be used to build the expert systems, and also is suitable to build EDS. Our approach is to take advantage of an expert system loosely-coupled with a database system. This experimental prototype was developed by using C-Prolog, C with embedded SQL under VMS, and RDB. With the loosely coupling approach, an interface was built between Expert Shell and RDB, which preserves their individuality. The Inference Engine (Expert Shell) provides various predicates that perform indirectly SQL query with RDB. "Indirectly" means this query is handled by an interface mechanism (IM)
that is designed by using C with embedded SQL. The demo of the approach is called "A Credit Evaluation Expert Database System".

**Background Perspective**

The Credit Evaluation Expert Database System is a prototype of EDS. This example comes from the world of banking (an imaginary bank) to evaluate requests for credit from small business ventures. We give a fictionalized account of the development of a simple expert database system for evaluating client credit requests from a bank. The bank considers three factors that are of the most importance in considering a request for credit from a "client". Clients here refer to small business ventures.

The most important factor is the collateral that can be offered by the client in case of a venture. The various types of collateral are divided into categories. Current deposits, whether local or foreign, are first-class collateral. Stocks are examples of second-class collateral, while the collateral provided by mortgages and the like is regarded as illiquid. Also very important is the client's financial record. Experience in the bank has shown that the two most important factors are the client's net worth per assets, and the current gross profits on sales. The client's short-term debt per annual sales should be considered in evaluating the client's record.

The remaining factor to be considered is the expected yield to the bank. This is a problem that banks have been working on for a while. Programs exist
to give the yield of a particular client profile. We can thus assume that the information will be available in the desired form.

A bank uses qualitative terms in speaking about these three factors: "The client had an excellent financial rating, or a good form of collateral. His venture would provide a reasonable yield, etc."

After talking to the bank, it became clear that a significant amount of the expert knowledge they described could be naturally expressed as a mixture of procedures and rules. Bank given rules for determining ratings for collateral, and financial rating. These involved considerable calculations, and in fact the bank admitted that to save work in the long term, they do a quick initial screen to see if the client was at all suitable.

This information is sufficient to build a small prototype of EDS.

**The System Architecture**

The current design of CEEDS makes two assumptions:

1. The evaluation rules of the bank are not affected by the update of the database schema.

2. The order of access to tuples of database predicates is not important.

CEEDS has been designed with the following goals in mind:

1. Although the semantics of Prolog sometimes have undesirable effects, programming in CEEDS should be the same as programming in Prolog. Hence, independently of how CEEDS is implemented, the impression given to the user...
should be that of Prolog.

2. The existence of a database system underneath Prolog should be transparent to the user. Whether something is stored in the virtual memory of Prolog or on disk should not affect any Prolog programs.

3. Both Prolog and the database system should be used unchanged, or else with minimal changes that in no way alter their basic features or the philosophy of their design.

The architecture of CEEDS, motivated by the above goals, is shown in Figure 1.

The description of the various is as follows:

**CREDIT.PRO** is an expert shell written in C-Prolog. It plays an expert role in the CEEDS. There two predicates in this program that are called IM.SC and DBM.SC respectively. The expert shell processes communicate with the database system through three VMS files called amount_kdb.pro, value_kdb.pro, and psyl.pro. Whenever the expert shell processes a query from a user, it calls IM.SC (Interface Mechanism) to translate user’s information in the format of facts in the knowledge base, which are stored in the above three files. For example, the file spy1.pro contains: spy1_client(ok) or spy1_client(bad) that tells the expert shell whether IM.SC has found the client’s information or not. The contents of the amount_kdb.pro and the value_kdb.pro are described below:

**IM.SC** plays a role of interface between the expert shell and the database management system. It is invoked by the expert shell. It fetches client's information
Figure 1. The Architecture of CES.

from RDB, and produced three files: spy1.pro, amount_kdb.pro, and value_kdb.pro. For example, some specific contents of the file amount_kdb.pro are listed as follows:

amount(local_currency_deposits, client1, 30000).

amount(foreign_currency_deposits, client1, 20000).

amount(bank_guarantees, client1, 3000).

amount(negotiate_instruments, client1, 5000).
amount(stock, client1, 9000).
amount(mortgage, client1, 12000).
amount(documents, client1, 14000).

Specific contents of file value_kdb.pro are listed below:
value(net_worth_per_assets, client1, 40).
value(last_year_sales_growth, client1, 20).
value(gross_profits_on_sales, client1, 45).
value(short_term_debt_per_annual_sales, client1, 8).

DBMS.SC plays the role of a database manager. It performs four basic functions: select, insert, update, and delete. The source code of this sample can be found in Appendix A and Appendix B.
CHAPTER IV
IASCE: AN INTELLIGENT ASSISTANT TO SOFTWARE COST ESTIMATION

It has long been a belief of the artificial intelligence research community that some AI techniques can be successfully applied to automatic programming. A paper by Frenkel [2] surveys research efforts in this area and provides a perspective on the progress that can be expected. The goal of his paper is to describe the development of an "expert" system to generate a design and high-level language instructions from a complete set of specification and user requirements. This is in the same spirit as developing a compiler to generate machine code from a high-level language. A typical expert system contains a knowledge base. When applied to software development, we can envision a series of knowledge-based software assistants for each of several phases of software life cycle, including project management, requirements analysis, implementation, validation and testing, and maintenance. It is doubtful that expert systems for automatic program generation will be successfully developed in the near future that will have a significant impact on software productivity. However, it is reasonable to expect that some automatic processes on the SE high level phases such as project management, requirements analysis will definitely improve software productivity.
In recent years, great efforts have been placed on the integration of AI and SE [3,4,5,6,7,8]. In fact, these efforts have lead to the design of a new class of systems, called Computer Aid Software Engineering (CASE), which use AI techniques to support software engineering. In this new field, expert systems play a significant role. At our knowledge, some CASE tools have been successfully used currently in the software industry [6,7]. Some new CASE tools will be developed to assist certain SE activities such as project management. A paper "Assisting Managers in Project Definition: Foundations for Intelligent Decision Support", written by Kent D. Bimson and Linda Boehm Burris [8], describes how to apply AI technique into SE project management activities. However, the authors didn’t mention a very important activity in the project management - software cost estimation. At our knowledge, none of the automatic CASE tool has been developed to assistant this very important field of SE.

There are several reasons why none of CASE tool has been made to assist the software cost estimation (SCE) activity: (a) lack of accurate data to build the knowledge base because it is difficult to collect real data from the software industry companies; (b) as we mentioned on Chapter I, there is no universal software cost estimation model in existance today. To solve these problems, however, we have to build the expert system at first in order to collect real data from the software industry. Some existing SCE models such COCOMO, Doty Model, PRICE S model can be considered as conceptual experts to build a "real expertise". If we develop such a system now, we will expect a real SCE tool to appear in the near
future, since such a system will perform self-learning and it will automatically improve the expertise based on data accumulation.

Thus, based upon many years of analyzing software cost models, which have been done by some dedicated scientists in universities and industry, a CASE tool called IASCE has been proposed to be built in order to assist the project manager to estimate a proposed project cost. This tool collects accurate data from current software industry so that researchers may explore new software models and metrics for software cost estimation. IASCE also supports multiple software cost estimation models and their corresponding metrics, tractable for project management control, feedback and learning activities. In addition, IASCE provides for the establishment of project specific cost models and corporate metrics for the models, enables tracing of these models and metrics throughout the software life cycle via feedback and post mortem evaluation, and offers a mechanism for long range improvements of software cost estimation. It can be tailored to a specific software development environment according to the requirements of a certain client.

In this chapter, first we present the design goal of IASCE, then we describe the system architecture in order to meet the requirement of our design goal. Since the tools and techniques needed to build the architecture of EDS have been presented in Chapter III, we do not repeat the same information in this Chapter. However, EDS architectures for the IASCE systems are presented below:

We have implemented the first IASCE prototype in Prolog, C, SQL on the
WMUnet under the VMS operating system. The DBMS in a DEC relational database management system is called RDB. In the initial implementation, the system is composed of a stand alone COCOMO model. The experimental results of the prototype system verified the feasibility of the goals set by the IASCE system.

This chapter is organized in four sections as follows: (1) a description of the design goal of the IASCE system, (2) a preview of the architecture of IASCE systems, (3) details of techniques used to build such architecture, and (4) a description of the first IASCE prototype and its current status.

**Design Goal of the IASCE System**

The IASCE system automates as much of the cost estimating process as possible, by providing validation, evaluation, and analysis of the cost estimating process, creating and using historical data and knowledge bases that incorporate experience from both prior projects and experienced project managers. IASCE will automate these cost estimating aspects within the framework of the analytical dimension of the software cost estimating models presented in Chapter II. In this section we present the requirements for IASCE, its architecture, and the scope of the first experimental prototype. We have planned a series of prototypes being built using the interactive enhancement model. This approach is necessary because more research is needed in many areas (e.g., software cost estimation,
artificial intelligence, CASE, expert database systems) before the idealized IASCE system can be built which will fulfill the entire set of design goal of the IASCE system.

The requirements for the IASCE system can be derived from the previous chapter in a natural way. These requirements can be divided into direct requirements (defined by and of obvious interest to the IASCE user) and indirect requirements (defined by the IASCE designer).

**A Mechanism for Learning**

The system support "learning mechanism" allows the user to define his/her own estimation model operationally and also to refine it into quantitative questions and metrics. The selection of the appropriate estimation model and its tailoring can be supported by IASCE. The user will either select an already existing estimation model without any changes or generate a new one. A new estimation model can be generated from scratch or by reusing pieces of existing estimation models. This process is done by an expert shell, which determines how to make an entire new estimation model or how to reuse parts of an existing model depending on the user's direction. The user will get on-line help to define his/her specific estimation model according to the estimation model definition which is given by the system. Based on this mechanism, IASCE system will search for an estimation model in the CASE data base. If no appropriate model exists, the user will be guided in developing one. Based on the tractability of models into a set
of the mode feature and cost-driver attributes the system will identify reusable pieces of existing models and compose as much of an initial model as possible. This initial model will be completed with user interaction.

For example, if a user wants to develop an estimation model for assessing an estimation method used in a particular environment, the system might compose an initial model by reusing pieces from an existing model, such as COCOMO, to assess some predicates in the same environment or from an estimation model for assessing the same system estimation method in a different environment. A complete estimation model includes rules for interpretation of metrics and guidelines for the collection of the prescribed data. As much of this information as possible will be generated by the IASCE system automatically.

A Mechanism for Producing a Variety of Reports

IASCE provides a variety of reports depending on different purposes. For example, the documents of estimation, evaluation, new model generation, and interpretation results in the form of hard copies are supported by the system. Project managers are interested in interpretation results, in the appropriate documentation of tables, histograms and other kinds of textual and graphical representations. The system automatically generates this kind of reports.

A User Friendly Interface

The system provides a physical and a logical user interface. The physical
user interface provides a menu or a command driven interface between the user and the IASCE system. Graphics and window mechanisms are incorporated whenever useful and possible. The logical user interface is the user’s view of software cost estimation and its model evaluation. Users are not allowed to directly access estimation tools. The access to estimation tools when using IASCE is only through an expert shell. IASCE enforces this top-down approach to estimation and evaluation via its logical interface.

**Model Evaluation**

For evaluating the accuracy of the model’s estimates, the system uses the IASCE data base of completed software projects, comparing the estimates on these projects given by the existing SCE models such as the COCOMO model, Doty model, Bailey-Basili meta-model, and other models with the actual efforts, schedules, and phase distributions experienced by the projects. For instance, Intermediate COCOMO has 17 variables (size, mode, and 15 cost driver attributes).

Evaluating a model with the actual efforts can be done by the following formula:

\[
\text{Percentage of relative error} = \frac{|\text{EST} - \text{ACT}|}{\text{ACT}}
\]

Evaluating a model with actual development schedule considers the features of the developed projects. The system gives a histogram report about percentage...
of relative error of accuracy for both traditional, single increment software development and incremental development projects.

Evaluating a model with actual phase distribution considers four phases throughout a product completion, that is, Product Design, Detailed Design, Code and Unit Test, and Integration and Testing.

**Tailoring a Model to a Particular Installation**

In general, the models presented in the expert system provide a reasonable estimation mechanism for most software cost estimation situations. However, a given installation can develop a specially calibrated and tailored version of a certain estimation model which will be more accurate and easy to use within the context of the particular installation. The expert system processes the following calibration and tailoring mechanisms:

1. Calibrating the model nominal effort equations to the installation’s experience.

2. Consolidating or eliminating redundant cost driver attributes within the model.

3. Adding further cost driver attributes which may be significant at this installation, but not in general; for example, security or privacy restrictions. Each of these considerations is discussed in more detail in [10].
Coupling an Expert System Into a DBMS

As mentioned above, the system is composed of a knowledge base of rules (server for models and their corresponding metrics), facts (domain of metrics) and an inference mechanism that uses the knowledge base to respond to queries posed by project managers. The objective of this system is to capture the knowledge of outstanding researchers and experienced project managers in the software cost estimation domains and make it generally available to nonexpert users. At present, the prototypical system focuses on narrow domains and has small knowledge bases, but once the system is expanded, the knowledge bases will become more difficult to manage and their major management activity will be in the access, update and control of the rules and facts. The system also needs to store more information of projects which will be used for evaluation.

All data and knowledge required to support estimation, evaluation, tailorability and tractability needs to be stored in the IASCE database. Such a data repository, needs to be able to store estimation models and all kinds of evaluation data. It needs to store data derived from current projects as well as historical data from prior projects. The effectiveness of such a data repository will be improved by learning and feedback.

Fortunately, a DBMS can allow multiple user access to the same collection of information for purposes of retrieval and update. The DBMS also provides a separate software facility required between the users and the database to protect
the shared information. This facility provides consistency, recovery, concurrence
and security control. In addition, a DBMS can provide other advantages in con­
trolling data redundancy and distribution, and in making application programs
easier to develop and maintain.

This requirement is especially important due to the fact that current data­
base technology is not suited to properly support software engineering concepts.
However, as we described in Chapter III, EDS uses a logic programming style for
formulating queries and constrains, and database technology for providing effi­
cient and reliable access to mass-memory. The goal of EDS is to couple an
Expert System into a DBMS. Therefore, EDS can solve this problem.

System Architecture

The IASCE architecture in Figure 2 describes the individual components of
the IASCE system and their interrelationships, according to the design goal pre­
sented in the first section of this chapter.

1. Graphics User Interface consists of the graphics user interface tools. The
main menu provides a "mouse-and-menu" interface to the expert system monitor.

2. Expert System monitor controls the execution of other IASCE modules
and awaits incoming requests. When a message arrives, it invokes a request inter­
pretation module to determine which module should execute next.

3. Project Data Entry Validation: The input of non-automatically collected
data and their validation is implemented by this tool. The data considered are all
the data related with evaluation of project cost model. This kind of information is collected into the project’s database.

4. Expert Judgment tool is consulting with one or more experts, who use their experience and understanding of the proposed project to arrive at an estimate of its cost. The strengths and weaknesses of these methods are highly

Figure 2. The Architecture of the IASCE System.
complementary to the strengths and weakness of algorithmic models.

5. Report Generator produces all kinds of reports, including the final cost estimation result. In addition, it produces the reports such as results of evaluation about a new model, including a decision from the system that informs the user whether this model is accepted or not.

6. Expert Shell1 contains the inference strategies and control that simulates experts' model processing; it manipulates the rules (model, metrics) and facts (metrics domain) in order to produce final cost estimation. It makes effective use of the knowledge stored in the IASCE knowledge base to infer new knowledge and also a schedule that decides the order in which the rules should be applied. In the user's view, it plays the role of an experienced project manager. The logical user interface is implemented by this tool.

7. Expert SHELL2 plays the role of an administrator for entry of both rules and facts. In other words, it provides a "learning mechanism" to accept the new SCE model and its corresponding metrics from an experienced project manager or a SCE researcher. The SCE model and metrics are presented by a set of rules and facts.

8. Model Evaluation provides a function for evaluation of a completed project, and also supports a mechanism for refinement of a software cost model. This module performs the evaluation according to a particular SEC model. In addition, the IASCE Model Evaluation needs to know the specific authorizations of the user in order to know which evaluation functions can be performed by this
particular user. Consequently it tailors a certain SCE model into a specific software development environment according the requirements of a certain client. The IASCE Model Evaluation also provides analysis functions, for example, telling the user whether certain measures can be computed based on the data (models, metrics, and completed projects) currently available in the IASCE database.

9. Model and Metrics interpreter (MMI) performs an interface function between the expert shell and a DBMS. It is booted by the Expert System monitor, either to begin or to end a transaction. In the beginning of a transaction, it translates a set of rules and facts from IASCE Database into the Knowledge Base, then it returns control to the expert system monitor. At the end of a transaction, the expert monitor calls it to translate the Knowledge Base into the IASCE Database. In other words, this module performs the functions which retrieve the SCE model and its corresponding metrics from the IASCE database, and inserts the SCE model and its corresponding metrics into the IASCE database. In the initial implementation, this part is written in Prolog and C with embedded SQL.

10. IASCE repository consists of a SCE database which contains the conceptual knowledge, the experimental knowledge, and a project database. The SCE database contains the facts and rules related to the software cost driver, all cost models, heuristics and ideas for solving problems in the specific domain. In the initial implementation, the first cost model stored in IASCE database is called
COntstructive Cost Model (COCOMO).

11. IASCE database Interface performs the DBMS functions which include insert, retrieve, update, and delete.

Implementation of the Initial IASCE Prototype

The first of a series of prototypes has been developed for supporting verification of our new approach in Software Cost Estimation. This first prototype implements some modules of the IASCE architecture such as Expert Shell1 (Model & Metrics Selection), Model and Metrics Interpreter. However, some of the architecture of IASCE have not implemented yet, such as Expert System Monitor, Project Data Entry Validation, and IASCE database Interface, since the remaining portion of the architecture of IASCE will take long time to develop and requires a large amount of computer resources. In addition, the implementation of some of modules such as Module Evaluation and Expert Shell2 (New Model & Metrics Generator) needs long-range verification, but it does not affect verification for the design goal of the IASCE systems. In fact, the result of the experiments of the initial implementation of the prototype shows that we can build such IASCE system to improve the SCE techniques.

The first prototype enables the user to select an existing SCE model by using a unique model name. Estimation sessions can be run using existing SCE models. However, no support for the interpretation is provided. Metric values are presented to the user according to the underlying model for his/her
interpretation. Results can be documented on a line printer. A general schema for a SCE data repository has been developed and implemented. The current implementation is based upon the relational database system from the DEC Corporation.

The first prototype is running on WMUnet under VAX/VMS. It is implemented in Prolog and C with embedded SQL. The Expert SHELL1 is designed and implemented based on the Prolog program XSHELL, written by Michael A. Covington, Donald Nute, and Andre Vellino [51]. In addition, the basic structure of the knowledge base is also designed based on the requirements of XSHELL. More research is needed before the idealized IASCE system can be built. Major areas of research include SCE, AI, EDS. As results become available, they can be integrated into an enhanced prototype.

This section is organized in five sections as follows: (1) a description of how to represent SCE Models and Metrics in a knowledge base; (2) a description of how to represent the control strategy in the Expert Shell; (3) a description of how to represent the SCE equation in the IASCE knowledge base and the IASCE database, and how to translate between knowledge base and database; (4) a sample to explain the inference engine of the IASCE system; and (5) a description of a database schema for the IASCE environment.

**Representation of Models and Metrics**

It has become clear that the success of expert systems is more a function
of the power of the representation and richness of the rule base than the choice of control schemes. Emergence of languages especially suited for expert system construction [28], has pushed control scheme technology far ahead of knowledge acquisition technology in expert systems. In its basic form, a knowledge base is a collection of rules designed through dialogues between a domain expert and a knowledge engineer or by domain experts themselves. Although such rules are physically independent pieces of knowledge, they are highly interrelated. Some rules are applicable only if others have been previously fired; several rules may need the same information in order to fire, or several rules may infer the same conclusion. These relations are implicitly stored in every part of the knowledge base. All features of a knowledge base are suited to represent Software Estimation Model and Metrics. An expert shell provides a Goal-Question-Answer, and it is especial suited for our approach. Basili's paper[50] proposed a goal/question/metric (G/Q/M) paradigm to support SE environment. Basili surveys the features of SE and proposed the G/Q/M paradigm which was successfully used in SE environment. Therefore, in order to build an expert system, first we should represent the model and the metrics in a knowledge base through the definition of rules and facts to be included in the knowledge base. The definitions is described as follows:

1. Goal - A Software Algorithmic Estimation Model;
2. Representation of Estimation Model - A set of rules used by the control strategy to derive a goal(An Estimation Model);
3. Representation of Estimation Metrics - A set of rules associated with a certain goal;

4. Domain of Metrics - A set of facts which are in the value range of a metric.

When we are given the definition of rules and facts in the IASCE knowledge base, we group these rules and facts into different domains according to the types of knowledge. Basically, IASCE’s knowledge base contains the following three types of knowledge:

1. Conceptual Knowledge: Knowledge about software cost estimation is based on some existing cost estimating models such as COCOMO, COPMO, and Jensen’s [1]. This knowledge consists of templates of software cost estimating models and cost metrics which were derived from a historical database.

2. Experienced Knowledge: The IASCE system accepts the expertise of cost estimation from experienced project managers and executes a tailoring and calibrating process based on the conceptual knowledge.

3. Improvement Knowledge: Knowledge derived from the comparison between conceptual and experienced knowledge made by a heuristic mechanism. Finally, it can be derived through a set of criteria for evaluating a software cost model after completing the project.

The three types of knowledge use the same representation of Models and Metrics except that they are stored in different tables of the IASCE database. For example, the user can select an existing SCE model and its corresponding
metrics, which is defined as conceptual knowledge, to estimate a proposed project. If the user is an experienced project manager or an SCE researcher, his/her SCE knowledge can be considered as experienced knowledge. Assume that the user is an experienced project manager, when he/she selects an existing SCE model to estimate an ongoing project, enters his/her new judgement to modify the existing SCE model to generate a modified SCE model. The IASCE system uses both models to estimate the project cost, and store these results of estimation into different tables in the IASCE database. When this process is completed, the IASCE system will run Model Evaluation to compare the results of estimation based on these two modules. Consequently, an improved model and its corresponding metrics would be produced by the IASCE system. This model and its corresponding metrics are considered as improvement knowledge.

**Meta-Rules for Representing the Control Strategy**

All three types of knowledge are represented by a combination of meta-rules, equations, and metric rules. The term meta-rule is used here to refer to meta-information in a production rule format, dealing with the content of object-level rules. Meta-rules offer a means of guiding a production system to apply the "best" candidate from a set of heuristic hypotheses. A hypothesis (i.e. a software development mode) is considered and the inference mechanism collects a group of rules which conclude the truth or falsity of this hypothesis. An example of a hypothesis initiating forward chaining, and the resulting software
development mode built by the inference mechanism is the following:

Hypothesis: Software development mode is Organic mode.

RuleX IF

1. Organizational understanding of product objectives is thorough,
2. Experience in working with related software systems is extensive,
3. Need for software conformance with pre-established requirements is basic,
4. Need for software conformance with external interface specifications is basic,
5. Concurrent development of associated new hardware and operational procedures is some,
6. Need for innovative data processing architectures, algorithms is minimal,
7. Premium on early completion is low, and
8. The estimation of the product size is less than 50 KDSI,

THEN possible software development mode is Organic Mode.

Representation of the Nominal Effort Equation

Knowledge based systems include rule based paradigms used in expert systems, but are tightly integrated with conventional analytic tools. One equation is often worth a thousand rules. For this project, a weakness is the complexity of the representation of various model’s equations, if we represent the equations using rules. For example, at the first experiment, we implemented the initial prototype by using Prolog. The description of the algorithm is listed as follows:
Nominal Effort Estimation - algorithm.

/* mode(Value) Value = 1, 2, 3
   * COCOMO Nominal Effort and Schedule Equations
   */

mode_formula(1) :- !, nl,nl,
    write('Enter estimation of KDSI:'),
    readnumber(KDSI),
    V is 3.2 *(KDSI) ^ 1.05,
    nl,
    assert(mm(V)),
    write('3.2 x '), write(KDSI), write(' ^1.05 = '),
    write(V),
    write(' man-months '),nl,nl,
    f_to_int(V),
    int(NEE),
    abolish(int,1),
    write('The nominal development effort for this project = '),
    write(NEE), write(' man-months *),nl,nl.

mode_formula(2) :- !, nl,nl,
    write('Enter estimation of KDSI:'),
    readnumber(KDSI),
    V is 3.0 *(KDSI) ^ 1.12,
    nl,
    write('3.0 x '), write(KDSI), write(' ^1.12 = '),
    write(V),
    write(' man-months '),nl,nl,
    assert(mm(V)),
    f_to_int(V),
    int(NEE),
    abolish(int,1),
    write('The nominal development effort for this project = '),
    write(NEE), write(' man-months *),nl,nl.

mode_formula(3) :- !,nl,nl,
    write('Enter estimation of KDSI:'),
    readnumber(KDSI),
    V is 2.8 * (KDSI) ^ 1.20,
    nl,
    write('2.8 x '), write(KDSI), write(' ^1.20 = '),
    write(V),
    write(' man-months '),nl,nl,
    assert(mm(V)),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
However, if the computation of the equation was done by using conventional C language, it would have made the much easier. Another important reason for using the conventional C language to compute Nominal Effort Estimation is the translation between knowledge base and traditional database. The strength of this method is that it makes it easier to store all knowledge into the traditional database. We can use a single procedure to represent various model equations. For example, we can give a general equation for various cost models as follows:

\[ MM = constant1 + constant2 \times KDSI^{exp1} \]

The parameters of the equation are shown in Table 12. A procedure written in C for this purpose is shown below.

Nominal Effort Estimation - algorithm.

```c
equation(constant1, constant2, exp1, KDSI)
float constant1, constant2, exp1;
int KDSI;
{
    int MM;
    MM = constant1 + constant2 * pow(KDSI, exp1);
    return(MM);
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 12

The Parameters of Equation of Various Cost Estimating Models

<table>
<thead>
<tr>
<th>Cost Estimation Models</th>
<th>exp1</th>
<th>constant2</th>
<th>constant1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walston-Felix</td>
<td>0.91</td>
<td>5.2</td>
<td>0</td>
</tr>
<tr>
<td>Bailey-Basili</td>
<td>1.16</td>
<td>0.73</td>
<td>5.5</td>
</tr>
<tr>
<td>Boehm-Mode1</td>
<td>1.05</td>
<td>3.2</td>
<td>0</td>
</tr>
<tr>
<td>Boehm-Mode2</td>
<td>1.12</td>
<td>3.0</td>
<td>0</td>
</tr>
<tr>
<td>Boehm-Mode3</td>
<td>1.20</td>
<td>2.8</td>
<td>0</td>
</tr>
<tr>
<td>Doty Model</td>
<td>1.047</td>
<td>5.288</td>
<td>0</td>
</tr>
</tbody>
</table>

Unlike rules, equations are often known and well understood. The parameters of the equation are stored in the knowledge base as well as in the IASCE database. This consideration is not only to make the representation of estimation models clean but also to make the installation easy. For instance, the organic mode development effort estimation equation of COCOMO is of the form

\[
MM_{DEV} = a_4 (KDSI)^{1.05} m(X)
\]  

(2.3)

The parameters \(a_4\) and \(m(X)\) change with the process of a specific installation and improvements based on the new knowledge derived by expert systems. Suppose that a specific company has completed four projects, and also collected data on five projects, whose sizes KDSI are 5, 10, 20, 30; whose overall effort adjustment factors \(m(X)\) are 0.75, 1.0, 0.80, 0.7; and whose actual development effort in MM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
are 15, 44, 60, 133. Then, the average value of $a_i (i=1,2,3,4)$ is the solution of the system of linear equations

$$15 = a_1 (5) \times 1.05 \times 0.75$$
$$44 = a_2 (10) \times 1.05 \times 1.00$$
$$60 = a_3 (20) \times 1.05 \times 0.80$$
$$133 = a_4 (40) \times 1.05 \times 0.70$$

The average value of $a_i$ is 3.7, thus the tailored effort equation for the installation is

$$MM_{ DEV} = 3.7 (KDSI)^{1.05}m(X) \quad (2.4)$$

The estimation produced by the estimator is

$$MM_{ EST} = 3.2 (KDSI)^{1.05}m(X) \quad (2.5)$$

We use equation (2.4) as the new estimating model for predicting new project cost. The term metric rules is considered as a measurement of attribute cost drivers.

**The Inference Engine of the IASCE System**

The inference engine of the IASCE system is a production system. The basic control cycle of a production system can be decomposed into two phases: recognition and action. The recognition phase can be further broken down into selection and conflict resolution stages. Selection is the process of examining rules to determine which consequent meets the current goal (i.e., cost estimation model), and subsequently collecting these rules in what is known as the conflict
set. The action phase is the computing of equations.

Following is a reasonably concise example of a cost estimating process, which shows how the expert system learns from the project manager.

Suppose we are estimating the cost to develop a set of batch COBOL programs to keep track of a company's stock. During the dialog with the project manager, the expert system can display questions and results of the consultation. The system has built-in questions and metrics which are stored in the IASCE knowledge base. However, an experienced project manager can, in addition, create his own questions and their corresponding metrics on a single screen. The expert system determines the characteristics that best fit the profile of an organic-mode project. Then, the project manager estimates that the project needs about 60 (KDSI) according to his/her previous experience and considers that only 10 of the existing cost drivers significantly effect the development effort. In addition, he/she decides that the size of the development team is very important, and gives a rating level and its corresponding numerical value. The composite multiplier is determined to be \( m(X) = 0.87 \) after executing the computing procedure. Then the IASCE system derives a set of heuristic results as follows:

\[
MM_{dev} = 3.7 \times (60)^{1.05} \times 0.87 = 237 \text{ (man-months)}
\]

If the personnel cost is $5000 per MM, then

\[
COST = [237 \times \text{MM}] \times \frac{\$5000}{\text{MM}} = \$1185K.
\]

The time of development is computed to be

\[
TDEV = 2.5 \times (237)^{0.38} = 20 \text{ (months)}
\]
IASCE searches all the knowledge that the expert system has and refines it by using the new knowledge.

IASCE is organized around the systematic use of a knowledge base containing rules that link cost data to estimate specific product cost. In several ways, the program exploits the development software mode and its corresponding cost drivers by the use of rules to express this scientific knowledge. The program obtains answers from a user by a simply backward chaining through its rules.

The problem solving activities that IASCE uses are: identify the problem, process data, generate questions, collect information, establish hypotheses, group and differentiate, pursue and test hypotheses, explore and refine, and make a decision. The system informs itself about particular cases by requesting information about specific cost. At each point, the question IASCE asks is determined by the ongoing analysis of all previous questions. The system checks every possible hypothesis, so search economy is not an issue.

IASCE uses backward chaining so that the questions seem focused. By sticking with the questions that fan out in the implied AND/OR tree beneath a particular hypothesized conclusion, the questions are guaranteed to stick to that hypothesis. A backward chaining system can jump around, collecting information relevant first to one hypothesis and then to another.

A Database Scheme for the IASCE Environment

The schema of the IASCE database is designed based on the designing goal
we mentioned earlier. A knowledge-based cost estimator for software cost, IASCE performs a project cost estimations using a knowledge base of software-estimating expertise by making acquisitions from both cost-estimating researchers and experiential project managers. Through the IASCE database, this expert system accesses a software cost estimating database, a project database. The schemes of the IASCE database are shown in Table 13 through Table 17.

IASCE uses software cost estimating models contained in the cost estimating database to estimate a proposed project's cost, and stores these estimates in the project management database. Once the projects have been completed, IASCE uses the information which obtained from the phases of the life cycle of developing software to evaluate software cost estimating model in turn to refine the cost models.

Table 13

<table>
<thead>
<tr>
<th>Model_ID</th>
<th>Model_Name</th>
<th>Author_Name</th>
<th>Created_Date</th>
<th>Accurate_Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COCOMO</td>
<td>Boehm</td>
<td>1981</td>
<td>70%</td>
</tr>
<tr>
<td>2</td>
<td>Modified COCOMO</td>
<td>Wang</td>
<td>1995</td>
<td>80%</td>
</tr>
<tr>
<td>3</td>
<td>COPMO</td>
<td>Conte</td>
<td>1986</td>
<td>72%</td>
</tr>
<tr>
<td>4</td>
<td>Bailey-Basili Mate Model</td>
<td>Baily, Basili</td>
<td>1984</td>
<td>69%</td>
</tr>
<tr>
<td>5</td>
<td>Doty Model</td>
<td>Doty Associate Inc</td>
<td>1979</td>
<td>50%</td>
</tr>
</tbody>
</table>
### Table 14

Global Logical View of Software Development Mode Relation

<table>
<thead>
<tr>
<th>Model_ID</th>
<th>Mode_Name</th>
<th>Feature_Question</th>
<th>Criteria</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>organic</td>
<td>Organizational understanding of product objectives</td>
<td>through</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>experience in working with related software systems</td>
<td>extensive</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Need for software conformance with pre-established requirements</td>
<td>basic</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Need for software conformance with external interface specifications</td>
<td>basic</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Concurrent development of associated new hardware and operational procedures</td>
<td>some</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Need for innovative data processing architectures, algorithms</td>
<td>minimal</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Premium on early completion</td>
<td>low</td>
<td>action</td>
</tr>
<tr>
<td>1</td>
<td>organic</td>
<td>Product size range 50 KDSI</td>
<td>less than</td>
<td>action</td>
</tr>
</tbody>
</table>

We use a sample program IM.sc which is listed in Appendix C to show how to translate the information of a Model and its corresponding question, and metrics into the knowledge base. This module is invoked by a special predicate in the Expert Monitor called system ("run im"). Let us look how this program works using an example. At first, it reads a file named model.inf to distinguish which model to retrieve. Assume the model is Boehm-Mode2. Based on the
### Table 15
Global Logical View of Model Equation Relation

<table>
<thead>
<tr>
<th>Model_ID</th>
<th>Mode_Name</th>
<th>Formulae_Type</th>
<th>exp</th>
<th>Constant1</th>
<th>Constant2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>organic</td>
<td>POW</td>
<td>1.05</td>
<td>0</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>semidetached</td>
<td>POW</td>
<td>1.12</td>
<td>0</td>
<td>3.0</td>
</tr>
<tr>
<td>1</td>
<td>embedded</td>
<td>POW</td>
<td>1.20</td>
<td>0</td>
<td>2.8</td>
</tr>
<tr>
<td>4</td>
<td>embedded</td>
<td>POW</td>
<td>1.16</td>
<td>5.5</td>
<td>0.73</td>
</tr>
<tr>
<td>5</td>
<td>embedded</td>
<td>POW</td>
<td>1.04</td>
<td>0</td>
<td>5.288</td>
</tr>
</tbody>
</table>

### Table 16
Global Logical View of Metric Relation

<table>
<thead>
<tr>
<th>Driver_Name</th>
<th>Metric_Name</th>
<th>Criteria</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>nominal</td>
<td>less than or equal 50% use of available execution time</td>
<td>1.00</td>
</tr>
<tr>
<td>TIME</td>
<td>high</td>
<td>70% use of available execution time</td>
<td>1.11</td>
</tr>
<tr>
<td>TIME</td>
<td>very_high</td>
<td>85% use of available execution time</td>
<td>1.30</td>
</tr>
<tr>
<td>TIME</td>
<td>extra_high</td>
<td>95% use of available execution time</td>
<td>1.35</td>
</tr>
<tr>
<td>AEXP</td>
<td>very_low</td>
<td>less than or equal to 4 months experience</td>
<td>1.66</td>
</tr>
<tr>
<td>AEXP</td>
<td>low</td>
<td>1 year experience</td>
<td>1.13</td>
</tr>
<tr>
<td>AEXP</td>
<td>nominal</td>
<td>3 years experience</td>
<td>1.00</td>
</tr>
<tr>
<td>AEXP</td>
<td>high</td>
<td>6 years experience</td>
<td>0.91</td>
</tr>
<tr>
<td>AEXP</td>
<td>very_high</td>
<td>12 years experience</td>
<td>0.82</td>
</tr>
</tbody>
</table>
IASCE database schema which is shown in Table 12, after running this program, a program for computing equation will be placed in the file named equation.c, which source code is shown as below:

```c
equation(KDSI)
int KDSI;
{
    FILE *fp, *fopen();
    int MM;
```
fp = fopen("norm_cost.pro", "w");
MM = 3.0 * pow (KDSI, 1.12);
fprintf(fp,"%d", MM);
return;
}

After executing predicate system ("run im"), Expert Monitor also calls a predicate
to produce an executable program named comp_equ.pro. The program comp_equ.pro is loaded whenever
Expert Shell needs it from mass-memory into main-memory to join the knowledge base.

As we mentioned in the previous sections, we need load rules and facts which represent the SCE model and its corresponding Metrics into main-memory to form the knowledge base. In [10], more examples show how to process a SCE procedure by using Tables 2.4, through 2.7. This was presented in Chapter II. Based on these tables, we designed a set of schemes to store the contents of these tables into the IASCE database. The samples of these schemes are shown in Tables 13 through 17. The schemes of these tables and their relationships are presented below.

Schema 1 gives a global logical view of software development mode relation. It can be presented as follows:

**Relational table name:** model_mode table

Attributes are Model_ID, Mode_Name, Feature_Question, Criteria, and Status.
This table is designed to store the contents of Table 4 into the IASCE database.
A sample of the contents of this schema is shown in Table 14.
**Relational table name: model equation table**

Attributes are Model_ID, Model_Name, Formulae_Type, exp, Constant1, and Constant2. This table is designed to produce a computing equation for a certain mode. A sample of the contents of this schema is shown in Table 15.

**Relational table name: cost driver table**

Attributes are Model_ID, Driver_Name, Driver_Question, and Status. This table is designed to store Table 5 into the IASCE database. A sample of the contents of this table is shown in Table 17.

**Relational table name: metric table**

Attributes are Driver_Name, Metric_Name, Criteria, and Rating. This table is designed to store Table 6 into the IASCE database. A sample of the contents of this table is shown in Table 16.

In the initial prototype, we implemented these schemes. We have the same way as translating a model equation from the IASCE database into the main-memory knowledge base. In other words, we used a set of predicates and their corresponding programs to process the communication between the Expert Shell1 and the IASCE database. A sample of knowledge base that is produced by these means is shown in Appendix D. The structure of the knowledge base is described in the book [51], written by Michael A. Covington, Donald Nute, and Andre Vellino. A sample of the Expert Shell1 is shown in the Appendix E. This program was developed by modification of an existing XSHELL program, which was designed by Covington [51]. A sample transaction of the IASCE prototype is listed in Appendix F.
CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

Summary and Conclusions

Based on studying Software Cost Estimation (SCE) methodology, Expert Database System (EDS) architecture, and Computer Aided Software Engineering (CASE) tools, we have proposed a new approach to solve the SCE problem. This new approach is to develop a CASE tool called Intelligent Assistant to Software Cost Estimation (IASCE), which supports SCE activities of SE life cycle. The idealized IASCE collects accurate data from current software industry so that researchers may explore new software models and metrics for the SCE field. The idealized IASCE also supports multiple software estimation models and their corresponding metrics, tractable for project management control, feedback and learning activities. In addition, the idealized IASCE provides for establishment of project specific cost models and corporate metrics for the models, enables tracing of these models and metrics throughout the software life cycle via feedback and post mortem evaluation, and offers a mechanism for long range improvements of SCE. It also can be tailored to a specific software development environment based on the certain requirements of the client.

Based on the definition of the idealized IASCE system, the architectures,
techniques and tools needed to build such a systems are presented. The very important architecture and technique needed to build the IASCE system is EDS. A loosely coupled architecture of EDS was used to build the first prototype of the IASCE systems. The representation of Models and Metrics was discussed, and the communication between Expert System and DBMS also was described.

The benefits of IASCE technology are varied. A well-defined software cost estimation model can be found through a long time range (4-7 years) use of IASCE technology. Further, this tool can help the project managers to avoid the frequent misinterpretations, underestimates and over-expectations. A related benefit of IASCE technology is that it provides a powerful set of insights on how a software organization can improve its productivity. Finally, IASCE technology provides an essential foundation for software project planning and control.

In the initial implementation, the system is composed of a standalone COCOMO model. This prototype was implemented using Prolog and C with embedded SQL on VAX/VMS operating system and RDBMS environment. The experimental results of the prototype system verified the feasibility of the goals of this thesis.

Although, it is doubtful that expert systems for automatic software cost estimation will be successfully developed in the near future nevertheless it will have a significant impact on software productivity. It is reasonable, however, to expect that some restricted forms of these may be available soon to assist project managers in software cost estimation. We believe that this proposed approach to SCE
can be used to build a real CASE tool which will be used by the project managers in a specific environment.

Future Research

The IASCE system is an ambitious project. It is assumed it will evolve over time and that we will learn a great deal from formalizing the various aspects of the IASCE system as well as integrating the various models. More research is needed before the idealized IASCE can be built. Major areas of study should include Software Cost Estimation methodology, Expert Database systems, and Artificial Intelligence. Specific high-priority topics are an Expert Database System for storing Cost Estimating key models, the definition of more and better cost models, mechanisms for better tailoring and reusing project knowledge, mechanisms for better interpreting metrics in the context of questions, better mechanisms for data access control and configuration management control, software engineering database definitions, and distributed system architectures. As results become available, they should be integrated into an enhanced prototype.

In the initial prototype, the system only implemented basic requirements of the IASCE system, and is composed of a stand alone COCOMO model. Enhancing the initial system can be done by storing more SCE models such COPMO, Doty Model, Bailey-Basili Meta-Model, PCA Price and other models into the IASCE database based on the scheme we presented in Chapter IV. The Expert Shell 2 in the IASCE architecture has not yet been implemented because more
research is needed on how to build such an idealized Expert Shell2. Major areas of study are how to make the system to automatically learn a new model and its corresponding metrics. Also, a graphics user interface (GUI) environment and an InterNet network environment is needed to be established in order to allow the SCE researchers to access the IASCE database. In this case, a security issue has to be considered for the idealized IASCE systems.

In fact, the idealized IASCE system can not be built without financial support and major effort. The financial support is needed to research and build particular tools or subsystems.
Appendix A

A Sample of Expert Database System
A Expert Database System for Credit Evaluation (Prolog Part)

Author: Shouli Wang
Create Date: March 17, 1992
Modified History: April 1, 1992

Description: This program is an Expert System part written by using C-prolog. There are two predicates as follows:

1. system("run im") - It calls a program named written by using C with embedded SQL which performs the transformation of a client's bank records. It plays a role of interface mechanism between an expert system and a traditional database (here is RDB). It fetches client's data from RDB, and translates data to facts of Knowledge Base. For example, if data fetched from RDB is:

   
   client = client1
   local_currency_deposits = 30000


then the program im.c translates this data to facts:

   amount(local_currency_deposits, client1, 30000).

credit.pro also load two files that record all facts translated by im.sc program. Two files are amount.pro and value.pro.

2. system("run dbms") - This predicate call a program written by using C with embedded SQL, which perform a role of database management system. The program named dbms.sc provides 4 functions for bank's teller.

   (1) . Search client's data,
   (2) Update client's data,
   (3) Insert new client's record,
   (4) Delete client's record.

After system loading client's record, this program starts evaluation. It plays an expert role in the bank.

start :-
  .consult('writeln.pro'),
   .consult('readstr.pro'),
   .consult('readnum.pro').

Welcome to the Credit Evaluation System
In order to dialogue with the system, you just enter a customer name and requested credit.
the system will automatically give you an report!
about the decision made by the system!
menu_select.
end_program.
end_program :- !.

menu_select :-
  write('---------------------------------------------------------------'), nl,
  write('1. query about credit  2. modify database  3. exit from expert database system'), nl,
  write('---------------------------------------------------------------'), nl,
  write('Enter a number choice ----------------->'), nl, nl,
  read_number(Choice), nl,
  process(Choice).
  process(3) :- !,
end_program.

process(1) :- !,
  put(1), nl,
  system("run im"),
  consult('spy1.pro'),
  spy1_client(Spy),
  ok_profile1(Spy).

process(2) :- !,
  system("run dbms"),
  menu_select.

ok_profile1(Spy) :- Spy == ok, !,
  write('OK'), nl,
  profile(Client),
  credit(Client, Answer),
  abolish(amount, 3),
  abolish(value, 3),
  abolish(spy1_client, 1),
  abolish(requested_credit, 2),
  abolish(bank_yield, 2),
  menu_select.

ok_profile1(Spy) :- Spy == bad, !,
  write('BAD'), nl,
  abolish(spy1_client, 1),
  menu_select.

/* Credit Evaluation:  
credit(Client, Answer) <- Answer is the reply to a request by Client for credit. */

credit(Client, Answer) :- nl, nl,
  reconsult('amount_kdb.pro'),
  reconsult('value_kdb.pro'),
  nl, nl, nl,
  collateral_rating(Client, CollateralRating),
  financial_rating(Client, FinancialRating),
  bank_yield(Client, Yield),
  evaluate(profile(CollateralRating, FinancialRating, Yield), Answer),
  report(Answer, Report),
  writeln(Report).

/* The collateral rating module  */
collateral_rating(Client, Rating) :-
  collateral_profile(Client, FirstClass, SecondClass, Illiquid),
  /* Other code for collateral_rating */
collateral_evaluation(FirstClass,SecondClass,Illiquid,Rating).
collateral_profile(Client,FirstClass,SecondClass,Illiquid) :-
  requested_credit(Client,Credit),
collateral_percent(first_class, Client,Credit,FirstClass),
collateral_percent(second\_class,Client,Credit,SecondClass),
collateral_percent(Illiquid,Client,Credit,Illiquid).
collateral_percent(Type,Client,Total,Value) :-
  setof(X,Collateral ^ (collateral(Collateral,Type),
    count(Collateral,Client,X)),Xs),
  sumlist(Xs,Sum),
  Value is Sum * 100/Total.
/* Evaluation rules */
collateral_evaluation(FirstClass,SecondClass,Illiquid,excellent)
  :- FirstClass >= 100.
collateral_evaluation(FirstClass,SecondClass,Illiquid,excellent)
  :- FirstClass > 70, FirstClass + SecondClass >= 100.
collateral_evaluation(FirstClass,SecondClass,Illiquid,good)
  :- FirstClass + SecondClass > 60,
    FirstClass + SecondClass < 70,
    FirstClass + SecondClass + Illiquid >= 100.
collateral_evaluation(FirstClass,SecondClass,Illiquid,good)
  :- FirstClass + SecondClass > 60,
    FirstClass + SecondClass < 90.
collateral_evaluation(FirstClass,SecondClass,Illiquid,moderate)
  :- FirstClass + SecondClass =< 60.
/* Financial rating */
/* financial_rating(Client,Rating) <- Rating is a qualitative
description assessing the financial record offered by Client to
support the request for credit. */
financial_rating(Client,Rating)
  :- financial_factors(Factors),
    score(Factors,Client,0,Score),
    calibrate(Score,Rating).
/* Financial evaluation rules */
calibrate(Score,bad) :- Score =< -500.
calibrate(Score,medium) :- -500 < Score, Score < 150.
calibrate(Score,good) :- 150 =< Score, Score < 1000.
calibrate(Score,excellent) :- Score$ >= 1000.
/* weighting factors */
score([(Factor,Weight)|Factors],Client,Acc,Score)
  :- value(Factor,Client,Value),
    Acc is Acc + Weight * Value,
    score(Factors,Client,Acc1,Score).
score([],Client,Score,Score).
/* Final evaluation */
/* evaluate(Profile, Outcome) <- Outcome is the reply to the
client's Profile. */
evaluate(Profile,Answer) :-
  rule(Conditions,Answer),verify(Conditions,Profile).
verify([condition(Type,Test,Rating)|Conditions],Profile)
  :-
    scale(Type,Scale),
    select_value(Type,Profile,Fact),
compare(Test, Scale, Fact, Rating),
verify(Conditions, Profile).
verify([], Profile).
compare(='=', Scale, Rating, Rating).
compare('>', Scale, Rating1, Rating2) :-
    precedes(Scale, Rating1, Rating2).
compare('$$>=$$', Scale, Rating1, Rating2) :-
    precedes(Scale, Rating1, Rating2); Rating1 == Rating2.
compare('>', Scale, Rating1, Rating2) :-
    precedes(Scale, Rating2, Rating1).
compare('$$<=$$', Scale, Rating1, Rating2) :-
    precedes(Scale, Rating2, Rating1); Rating1 == Rating2.
predecedes([R1|Rs], R1, R2) :-
    R1 == R2, precedes(Rs, R1, R2).
select_value(collateral, profile(C, F, Y), C).
select_value(finance, profile(C, F, Y), F).
select_value(yield, profile(C, F, Y), Y).
/* Utilities */
sumlist([], Sum) :-
    sumlist(Is, Issum), Sum is I + Issum.
sumlist([1], 0).
/* Bank data and rules */
collateral(local_currency_deposits, first_class).
collateral(foreign_currency_deposits, first_class).
collateral(negotiate_instruments, second_class).
collateral(mortgage, illiquid).
financial_factors([net worth per assets, 5],
    last year sales growth, 1),
    (gross profits on sales, 5),
    (short term debt per annual sales, 2)).
rule([condition(collateral, '>=', excellent),
    condition(finance, '>=', good),
    condition(yield, '>=', reasonable)], give_credit).
rule([condition(collateral, '>=', excellent),
    condition(finance, '>=', good),
    condition(yield, '>=', reasonable)], consult_superior).
rule([condition(collateral, '<=', moderate),
    condition(finance, '<=', medium)],
    refuse_credit).
scale(collateral, [excellent, good, moderate]).
scale(finance, [excellent, good, medium, bad]).
scale(yield, [excellent, reasonable, poor]).
report(give_credit,
    ['--------------------- EXPERT SYSTEM REPORT ---------------------'],
    'WORLD BANK can issue the CREDIT to this client since
    his credit satisfies BANK requirement! ISSUE CREDIT '])
report(consult_superior,
    ['--------------------- EXPERT SYSTEM REPORT ---------------------'],
    'WORLD BANK can not issue the CREDIT to this client now since
    his credit is close to the requirement of BANK, but it is not
    stable. Therefore EXPERT suggest you to talk with manager
    CONSIDER ']).
report(refuse_credit,
['----------------- EXPERT SYSTEM REPORT ------------------',
'WORLD BANK can not issue the CREDIT to this client since
his credit does not meet BANK requirement! REFUSE CREDIT'])}.
Appendix B

A Sample Interface Mechanism
Interface Mechanism (Interface between Expert system and DBMS)
Author: Shouli Wang
Create date: March 20, 1992
Topic area: Expert Database System
Description: This program is called by an expert system named credit.pro. It fetches certain client's records from traditional database (here is RDB), and translates data to facts of Expert System's Knowledge Base. It plays a role of interface mechanism between an expert system and a traditional database. It translates a client's data into two files: 1. amount.pro, 2. value.pro.

#include <sysdef.h>
#include <stdio.h>
#include <descrip.h>
FILE *fp1, *fp2, *fp3, *fopen();
int client_id1;
char clientI[10];
int local_deposits1;
int foreign_deposits1;
int bank_guarantees1;
int negotiate_instruments1;
int stocks1;
int mortgage1;
int documents1;
int net_asset1;
int last_growth1;
int gross_sales1;
int short_sales1;
int end_of_file,found,choice;
int request_credit;
int total;
char c_name[10];
EXEC SQL INCLUDE SQLCA;
EXEC SQL DECLARE chasebank SCHEMA FOR FILENAME 'CHASE_BANK';
main()
{
    printf(" This program fetches the data from traditional database, translates the data to knowledge base. \n");
    printf(" Interface Mechanism program start...\n");
    printf(" Please enter the client name: \n");
    scanf("%s",c_name);
printf(" Please enter requested credit: \n");
scanf("%d", &request_credit);
search();
}

search()
{
EXEC SQL DECLARE A CURSOR FOR
SELECT CLIENT_ID, CLIENT, LOCAL_CURRENCY_DEPOSITS,
FOREIGN_CURRENCY_DEPOSITS, BANK_GUARANTEES,
NEGOTIATE_INSTRUMENTS, STOCKS, MORTGAGE, DOCUMENTS
FROM chasebank.AMOUNT
WHERE chasebank.AMOUNT.client = :c_name;
EXEC SQL DECLARE B CURSOR FOR
SELECT client_id, client,
NET_WORTH_PER_ASSETS,
LAST_YEAR_SALES_GROWTH,
GROSS_PROFITS_ON_SALES,
SHORT_TERM_SALES
FROM chasebank.VALUE
WHERE chasebank.VALUE.client = :c_name;
EXEC SQL OPEN A;
found = 0;
end_of_file = 0;
while((end_of_file == 0) && (found == 0)) {
EXEC SQL FETCH A INTO
:client_id1,
:client1,
:local_deposits1,
:foreign_deposits1,
:bank_guarantees1,
:negotiate_instruments1,
:stocks1,
:mortgage1,
:documents1;
if(SQLCA.SQLCODE == 100)
end_of_file = 1;
else if(*c_name == *client1) {
found = 1;
printf("client's name : %s\n", client1);
}
}
if(found == 1) {
fp1 = fopen("amount_kdb.pro","w");
fprintf(fp1,"amount(local_currency_deposits, ");
fprintf(fp1,"%s,%d).\n",client1,local_deposits1);
fprintf(fp1,"amount(foreign_currency_deposits, ");
fprintf(fp1,"%s,%d).\n",client1,foreign_deposits1);
fprintf(fp1,"amount(bank_guarantees, ");
fprintf(fp1,"%s,%d).\n",client1,bank_guarantees1);
fprintf(fp1,"amount(negotiate_instruments, ");
fprintf(fp1,"%s,%d).\n",client1,negotiate_instruments1);
fprintf(fp1,"amount(stocks, ");
EXEC SQL CLOSE A;
EXEC SQL OPEN B;
end_of_file = 0;
while(end_of_file == 0) {
    EXEC SQL FETCH B INTO
        :client_id1,
        :client1,
        :net_asset1,
        :last_growth1,
        :gross_sales1,
        :short_sales1;
    if(SQLCA.SQLCODE == 100){
        end_of_file = 1;
    } else {
        fp2 = fopen("value_kdb.pro","w");
        fprintf(fp2,"value(net_worth_per_assets,"");
        fprintf(fp2,"%s,%d\n",client1,net_asset1);
        fprintf(fp2,"value(last_year_sales_growth,"");
        fprintf(fp2,"%s,%d\n",client1,last_growth1);
        fprintf(fp2,"value(gross_profits_on_sales,"");
        fprintf(fp2,"%s,%d\n",client1,gross_sales1);
        fprintf(fp2,"value(short_term_debt_per_annual_sales,");
        fprintf(fp2,"%s,%d\n",client1,short_sales1);
        fprintf(fp2,"requested_credit("");
        fprintf(fp2,"%s,%d\n",client1,requested_credit);
        total = local_deposits1 + foreign_deposits1 + bank_guarantees1+ negotiate_instruments1 + stocks1 + mortgage1 + documents1;
        if( total > 50000 ) {
            fprintf(fp2,"bank_yield(");
            fprintf(fp2,"%s,reasonable).\n",client1);
        }
        fp3 = fopen("spy1.pro","w");
        fprintf(fp3,"spy1_client(ok).\n");
        fprintf(fp3,"profile(\n",client1);
    }
} EXEC SQL CLOSE B;
} else {
    fp3 = fopen("spy1.pro","w");
    printf("This client do not exist in the bank database !\n");
    fprintf(fp3,"spy1_client(bad).\n");
} return;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Appendix C

Sample: IM.SC
# include <sysdef.h>
# include <stdio.h>
# include <descrip.h>
FILE *fp, *fopen();
char m_name[20]; /* estimate model name, i.e. COCOMO */
char x;
char word[20];
char mode[20];
char feature[8][20];
char level[8][20];
char driver[15][20];
char rate[15][20];
char const1[20], const2, expl;
char f_type[20];

MMI_INITIAL()
{
    fp = fopen("model.inf", "r");
    /* i.e. there is a form "model(COCOMO)". */
    while ((x = fgetc(fp)) != EOF) {
        switch (x) {
            case '1':
                if (!strcmp(word, "model")) s1 = 1;
                for (i = 0; i <= 20; i++) word[i] = 0;
                k = 0;
                break;
            case '1':
                for (i = 0; i <= 20; i++) m_name[i] = word[i];
                k = 0;
                break;
            default:
                if (s1 == 1) { word[k] = x; k++;
                break;
        }
    }
    fclose(fp);
    EXEC SQL INCLUDE SQLCA;
    EXEC SQL DECLARE estimating_methods SCHEMA
    FOR FILENAME 'ESTIMATE_MODELS';
    EXEC SQL DECLARE A CURSOR FOR
    SELECT Mode_Name,
        Formulae_Type,
Exp,
Constant1,
Constant2,
FROM ee_estimating_methods.Model_Equation,
WHERE em.Model_ID = em.Model_ID
and em.Model_Name = : m_name;
EXEC SQL OPEN A;
found = 0;
end_of_file = 0;
while((end_of_file == 0) && (found == 0)) {
    EXEC SQL FETCH A INTO
        : m_name,
        : f_type,
        : expl,
        : const1,
        : const2;
    if(SQLCA.SQLCODE == 100)
        end_of_file = 1;
    else if(*m_name == Model_Name) {
        found = 1;
    }
}
if(found == 1) {
    fp1 = fopen("equation_kdb.pro", "w");
    fprintf(fp1,"equation(KDSI)\n");
    fprintf(fp1,"int KDSI;\n");
    fprintf(fp1,"int MM;\n");
    if(!strcmp(m_name,"pow")) {
        fprintf(fp1,"MM = %4.2f + %4.2f * pow(KDSI,%4.2f);\n");
        fprintf(fp1,"return (MM);\n");
        fprintf(fp1,"\n");
    }
}
Appendix D

IASCE-COCOMO: A Sample Shell Knowledge Base
/***************************************************************************/
* Program Name: IASCE-COCOMO.PRO 
* 
* A knowledge base of COCOMO model which contains following 
* data format: 
* 
* 1. An introductory statement. 
* 3. kb_unique(yes) or kb_unique(no). 
* 4. kb_explain(yes) or kb_explain(no). 
* 5. A set of identification rules. 
* 6. A set of questions for the parameters, drivers, modules used 
* in identification rules. 
* 7. Three equations for Nominal Estimation Effort. 
* 
* DEMO for An Intelligent Assistant for Estimation of Software 
* Product Cost. 
* Version: 1.0 Create Date: Feb. 28, 1992 
* Modify History: March 1-5, 1992 
* Author: Shouli Wang 
* The structure of this knowledge base is designed based upon 
* the sample knowledge CICHILD.PRO, written by Covington, Nute, 
* and Vellino[51]. 
* Non-commercial distribution of this file is permitted. 
* IASCE-COCOMO.PRO - A COCOMO knowledge base that contains a 
* SHELL1 knowledge base. Requires all procedures in 
* SHELL1.PRO. 
***************************************************************************/

:- ( clause(shell1,_); consult('shell1.pro')). 
:- ( clause(mathmenu,_); consult('mathmenu.pro')). 
:- ( clause(kem,_); consult('kem.pro')).

/* Any clauses for the predicates KB_INTRO, KB_REPORT, KB_UNIQUE, 
KB_EXPLAIN, KB_IDENTIFY, and KB_QUESTION should be removed from 
the knowledge base. */

:- abolish(kb_intro,1). 
:- abolish(kb_report,1). /* Use RETRACTALL instead */
:- abolish(kb_unique,1). /* of ABOLISH in some */
:- abolish(kb_explain,1). /* implementations */
:- abolish(kb_identify,1). 
:- abolish(kb_question,2).

kb_intro( 
[DEMO for IASCE - An Intelligent Assistant for Software Cost', 
'Estimation Version 1.0', 
'An Expert System for Estimation of Software Product Cost '],

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
'To use the program, simply describe the software development',
'mode by answering the following questions.'])

kb_report('Possible Nominal Effort Estimation: ')
kb_unique(no).
kb_explain(yes).
driver_feature(g,general).
driver_feature(c,considerable).
driver_feature(m,moderate).
driver_feature(t,through).
driver_feature(b,basic).
driver_feature(f,full).
driver_feature(e,extensive).
driver_feature(s,some).
driver_feature(mi,minimal).
driver_feature(l,low).
driver_feature(me,medium).
driver_feature(h,high).
driver_feature(1,50).
driver_feature(2,300).
driver_feature(3,all).
alt_rate(1,very_low).
alt_rate(2,low).
alt_rate(3,nominal).
alt_rate(4,high).
alt_rate(5,very_high).
alt_rate(6,extra_high).
modes(1,organic).
modes(2,semidetached).
modes(3,embedded).

/*
  kb_identify(Species)
  Each clause for this predicate provides a rule to be used with
  the utility predicates in the SHELL.PRO file to determine
  whether the feature to be identified is likely to belong to the
  specific mode.
*/
kb_identify([['
  'Step 1 - Nominal Effort Estimation has been done ! ''
  COCOMO Software Development is ORGANIC MODE ....''
  'i.e. Batch data reduction, scientific models, business models'
  'familiar OS, compiler, simple inventory, production control'
  ''
  'Step 2 - Determine Effort Multipliers has been done !'
  'Step 3 - Estimate Development Effort: We compute the''
  'estimated development effort for a project as the'
  'Nominal development', 'effort times the product of''
  'the effort multipliers for', '15 cost driver '
  'attributes. It is starting now....'
  '','']):-
  parm(feature1,c,t),
  parm(feature2,c,e),
parm(feature3,c,b),
parm(feature5,c,s),
parm(feature6,c,m),
parm(feature7,c,l),
calculate(feature8,n,1),

/* Software Cost Driver Ratings. */
  driver(rely,n,[1,2,3,4,5]),
  driver(data,n,[2,3,4,5]),
  module(cplx1,c,[y,n]),
  module(cplx2,c,[y,n]),
  module(cplx3,c,[y,n]),
  module(cplx4,c,[y,n]),
  module(cplx5,c,[y,n]),
  module(cplx6,c,[y,n]),
  driver(time,n,[3,4,5,6]),
  driver(stor,n,[3,4,5,6]),
  driver(virt,n,[2,3,4,5]),
  driver(turn,n,[2,3,4,5]),
  driver(acap,n,[1,2,3,4,5]),
  driver(aexp,n,[1,2,3,4,5]),
  driver(pcap,n,[1,2,3,4,5]),
  driver(vexp,n,[1,2,3,4]),
  driver(lexp,n,[1,2,3,4]),
  driver(modp,n,[1,2,3,4,5]),
  driver(tool,n,[1,2,3,4,5]),
  driver(sced,n,[1,2,3,4,5]).

kb_identify(['',
  'Step 1 - Nominal Effort Estimation has been done!
  'COCOMO Software Development is SEMIDETACHED MODE....
  'i.e. most transaction processing systems, new OS, DBMS
  'ambitious inventory, production control, simple command
  'control
  ''',
  'Step 2 - Determine Effort Multipliers has been done!
  'Step 3 - Estimate Development Effort: We compute
  'the estimated development effort for a project as the
  'Nominal development effort times the product of the effort
  'multipliers for the 15 cost driver attributes. It is
  'starting now....',']):-
  parm(feature1,c,c),
  parm(feature2,c,c),
  parm(feature3,c,c),
  parm(feature4,c,c),
  parm(feature5,c,m),
  parm(feature6,c,s),
  parm(feature7,c,m),
  calculate(feature8,n,2),

/* Software Cost Driver Ratings. */
  driver(rely,n,[1,2,3,4,5]),
  driver(data,n,[2,3,4,5]),
  module(cplx1,c,[y,n]),
  module(cplx2,c,[y,n]),
module(cplx3,c,[y,n]),
module(cplx4,c,[y,n]),
module(cplx5,c,[y,n]),
module(cplx6,c,[y,n]),
driver(time,n,[3,4,5,6]),
driver(stor,n,[3,4,5,6]),
driver(virt,n,[2,3,4,5]),
driver(turn,n,[2,3,4,5]),
driver(acap,n,[1,2,3,4,5]),
driver(aexp,n,[1,2,3,4,5]),
driver(pcap,n,[1,2,3,4,5]),
driver(vexp,n,[1,2,3,4]),
driver(lexp,n,[1,2,3,4]),
driver(modp,n,[1,2,3,4,5]),
driver(tool,n,[1,2,3,4,5]),
driver(scen,n,[1,2,3,4,5]).

kb_identify(,['1',
'Step 1 - Nominal Effort Estimation has been done !',
'COCOMO Software Development is EMBEDDED MODE....',
'i.e. large, complex transaction processing systems',
'ambitious, very OS, avionics, ambitious command-control',
',
'Step 2 - Determine Effort Multipliers has been done !',
'Step 3 - Estimate Development Effort: We compute',
'the estimated development effort for a project as ',
'the Nominal developmenteffort times the product of ',
'the effort multipliers for the 15 cost driver attributes.',
'It is starting now....', '']) :-
parm(feature1,c,g),
parm(feature2,c,m),
parm(feature3,c,f),
parm(feature4,c,f),
parm(feature5,c,e),
parm(feature6,c,c),
parm(feature7,c,h),
calculat(feature8,n,3),

/* Software Cost Driver Ratings.*/
driver(rely,n,[1,2,3,4,5]),
driver(data,n,[2,3,4,5]),
driver(time,n,[3,4,5,6]),
driver(stor,n,[3,4,5,6]),
driver(virt,n,[2,3,4,5]),
driver(turn,n,[2,3,4,5]),
driver(acap,n,[1,2,3,4,5]),
driver(aexp,n,[1,2,3,4,5]),
driver(pcap,n,[1,2,3,4,5]),
driver(vexp,n,[1,2,3,4]),
driver(lexp,n,[1,2,3,4]),
driver(modp,n,[1,2,3,4,5]),
driver(tool,n,[1,2,3,4,5]),
driver(scen,n,[1,2,3,4,5]).

kb_identify_type :-
module(cplx1,n,[1,2,3,4]),
assert(cost_rate(cplx,very_low,0.70)).

kb_identify_type :-
module(cplx2,n,[5,6,7,8]),
assert(cost_rate(cplx,low,0.85)).

kb_identify_type :-
module(cplx3,n,[9,10,11,12]),
assert(cost_rate(cplx,nominal,1.00)).

kb_identify_type :-
module(cplx4,n,[13,14,15,16]),
assert(cost_rate(cplx,high,1.15)).

kb_identify_type :-
module(cplx5,n,[17,18,19,20]),
assert(cost_rate(cplx,very_high,1.30)).

kb_identify_type :-
module(cplx6,n,[21,22,23,24]),
assert(cost_rate(cplx,extra_high,1,65)).

/*
kb_question(feature_or_Parameter,Question)
Each of these clauses provides a question or a simple menu to
be used by the utility predicates in the SHELL.PRO file to ask
the user whether the mode to be identified has the property or
what value the parameter takes for the mode.
*/

kb_question(feature1,
['Step1 - Nominal Effort Estimation: First, the following',
'8 questions are used to determine the project development',
'mode. There are three MODES',
'(1). ORGANIC (2). SEMIDETACHED (3). EMBEDDED',
each mode has a corresponding scaling equation which is',
used to determine the nominal development effort for',
the project in man-months as a function of the project',
size in KDSI(10000 delivered source instructions) and the',
project development mode. All these will be done by',
EXPERT SYSTEM... However, you need to answer the',
following questions; we assume you have',
a little bit project management experience. Good Luck!',
',
'Organizational understanding of product objectives?',
'(t) Thorough',
'(c) Considerable',
'(g) General.',
'Enter a letter in lower case, followed by a period:'])).

kb_question(feature2,
['Experience in working with related software systems?',
'(e) Extensive.',
'(c) Considerable.',
'(m) Moderate.',
'Enter a letter in lower case, followed by a period:')].

kb_question(feature3,
['Need for software conformance with per-established require',
ments?',

kb_question(feature4,
    ['Need for software conformance with external interface specifications?',
     '(b) Basic.',
     '(c) Considerable.',
     '(f) Full.',
     'Enter a letter in lower case, followed by a period:']).

kb_question(feature5,
    ['Concurrent development of associated new hardware and operational procedures?',
     '(s) Some.',
     '(m) Moderate.',
     '(e) Extensive.',
     'Enter a letter in lower case, followed by a period:']).

kb_question(feature6,
    ['Need for innovative data processing architectures, algorithms?',
     '(m) Minimal.',
     '(s) Some.',
     '(c) Considerable.',
     'Enter a letter in lower case, followed by a period:']).

kb_question(feature7,
    ['$Premium on early completion?',
     '(l) Low.',
     '(m) Medium.',
     '(h) High.',
     'Enter a letter in lower case, followed by a period:']).

kb_question(feature8,
    ['$Give your estimation of the Product size?',
     'i.e. the size of the product as 10,000 delivered source instructions (KDSI),',
     'enter 10, which means 10 KDSI.',
     '(1) less than 50 KDSI.',
     '(2) less than 300 KDSI.',
     '(3) All size.',
     'Enter a number, followed by a period.$]$.]

kb_question(rely,
    ['Step2 - Determine Effort Multipliers; Each of the 15 cost',
     'driver attributes in COCOMO has a rating scale and set of',
     'effort multipliers which indicate by how much the nominal',
     'effort estimate must be multiplied to account for the',
     'project having to work at its rating level for the attribute.',
     'These cost driver attributes and their corresponding',
     'effort multipliers are automatically given by EXPERT SYSTEM',
     'however you need answer the following question. We assume',
     'you know a little project management, otherwise check the',
     'manual of this EXPERT SYSTEM before you use it. Good Luck !',
     '
     'Product Attributes - Question:']},
(RELY) Required software reliability?,
(1). very low - Effect: slight inconvenience.,
(2). low - Low, easily recoverable losses.,
(3). Nominal - Moderate, recoverable losses.,
(4). High - High financial loss.,
(5). very high- Risk to human life.,

Enter a number.]).

kb_question(data,
['Product Attributes - Question:',
'(Data) Data base size ?',
(2). low - (DB bytes)/(Prog. DSI )<10.',
(3). Nominal - 10 <=(DB bytes)/(Prog. DSI)<100.',
(4). High - 100 <=(DB bytes)/(Prog. DSI) <1000.',
(5). very high- (DB bytes)/(Prog. DSI) >= 1000.',

Enter a number.]).

kb_question(time,
['Computer Attributes - Question:',
'(time) Execution time constraint ?',
(3) Nominal - <= 50% use of available execution time.',
(4) High - <= 70% use of available execution time.',
(5) Very high - <= 85% use of available execution time.',
(6) Extra High - <= 95% use of available execution time.',

Enter a number.]).

kb_question(stor,
['Computer Attributes - Question:',
'(stor) Main storage constraint ?',
(3) Nominal - <= 50% use of available storage.'
(4) High - <= 70% use of available storage.',
(5) Very high - <= 85% use of available storage.',
(6) Extra High - <= 95% use of available storage.',

Enter a number.]).

kb_question(virt,
['Computer Attributes - Question:',
'(virt) virtual machine volatility ?',
'virt means for a given software product, the underlying virtual',
'machine is the complex of hardware and software (Os, DBMS,etc.)',
'it calls on to accomplish its tasks.',
(2). Low - Major change every 12 months. ',
(3). Nominal - Major: 6 months; Minor: 2 weeks.',
(4). High - Major: 2 months; Minor: 1 weeks.',
(5). Very high - Major: 2 weeks; Minor: 1 days.',

Enter a number.]).

kb_question(turn,
['Computer Attributes - Question:',
'(turn) computer turnaround time ?',
(2) Low - Interactive. ',
(3) Nominal - Average turnaround time< 4 hours.',
(4) High - 4 hours <= Average turnaround time<12 hours.',
(5) Very high - Average turnaround time> 12 hours.',

Enter a number.]).
kb_question(acap,
['Personnel Attributes - Question:',
' (acap) Analyst capability ?',
'Team rating criteria: analysis (programming) ability, 
'efficiency, ability to communicate and cooperate.',
'(1) very low - 15th percentile.',
'(2) low - 35th percentile.',
'(3) Nominal - 55th percentile.',
'(4) High - 75th percentile.',
'(5) very high - 90th percentile.',
Enter a number.'])

kb_question(aexp,
['Personnel Attributes - Question:',
'(aexp) Application experience ?',
'(1) very low - <= 4 months experience.',
'(2) low - 1 year experience.',
'(3) Nominal - 3 years experience.',
'(4) High - 6 years experience.',
'(5) very high - 12 years experience.',
Enter a number.])

kb_question(pcap,
['Personnel Attributes - Question:',
'(pcap) programmer capability?',
'(1) very low - 15th percentile.',
'(2) low - 35th percentile.',
'(3) Nominal - 55th percentile.',
'(4) High - 75th percentile.',
'(5) very high - 90th percentile.',
Enter a number.])

kb_question(vexp,
['Personnel Attributes - Question:',
'(vexp) Virtual machine experience?',
'(1) very low - <= $1 month experience.',
'(2) low - 4 months experience.',
'(3) Nominal - 1 years experience.',
'(4) High - 3 years experience.',
Enter a number.])

kb_question(lexp,
['Personnel Attributes - Question:',
'(lexp) Programming language experience?',
'(1) very low - <= $1 month experience.',
'(2) low - 4 months experience.',
'(3) Nominal - 1 years experience.',
'(4) High - 3 years experience.',
Enter a number.])

kb_question(modp,
['Project Attributes - Question:',
'(modp) Use of modern programming practices?',
'(1) very low - No use.',
'(2) low - Beginning use.',
'(3) Nominal - Some use.',
'(4) High - General use.',

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(5). very high- Routine use.',
' Enter a number.'
])

kb_question(tool,
['Project Attributes - Question:',
'(tool) Use of software tools?',
'(1) very low - Basic microprocessor tools.',
'(2) low - Basic mini tools.',
'(3) Nominal - Basic mini/maxi tools.',
'(4) High - Strong maxi programming test tools.',
'(5) Very high- Add requirements, design, management',
'documentation tools. Enter a number.'
])

kb_question(sc ed,
['Project Attributes - Question:',
'(sc ed) Required development schedule?',
'(1). very low - 75% of nominal.',
'(2). low - 85% of nominal.',
'(3). Nominal - 100% of nominal.',
'(4). High - 130% of nominal.',
'(5). very high- 160% of nominal.',
' Enter a number.'
])

kb_question(c p lx 1,
['Type of Module: ',
'(1) Control Operations - Straightline code with a few nonnested',
'structured programming operators: DOS, CASES, IFTHENELSEs,',
'Simple predicates.',
'(2) Computational operations - Evaluation of simple expressions:',
'e.g. A = B + C * (D - E).',
'(3) Device-dependent Operations - Simple read, write statements',
'with simple formats.',
'(4) Data Management Operations - Simple arrays in main memory.',
'If your module meet one of above, enter a number otherwise ',
'a 0.'])

kb_question(c p lx 2,
['Type of Module: ',
'(5) Control Operations - Mostly simple nesting. Mostly simple predicates.',
'(6) Computational operations - Evaluation of moderate-level ',
'expressions: e.g.,',
'D = SQRT(B**2-4.*A*C).
'(7) Device-dependent Operations - No cognizance needed of ',
'particular processor or I/O device characteristics. I/O done ',
'at GET/PUT level. No cognizance of overlap. ',
'(8) Data Management Operations - Single file subsetting with ',
'no data structure; changes, no edits, no intermediate ',
'are files. If your module meet one of above, enter a number ',
'otherwise a 0.'])

kb_question(c p lx 3,
['Type of Module: ',
'(9) Control Operations - Mostly simple nesting. Some ',
'intermodule control. Decision tables.',
'(10) Computational operations - Use of standard math and ',
'statistical routines Basic matrix/vector operations.',
'(11) Device-dependent Operations - I/O processing includes ','

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
device selection status checking and error processing.

(12) Data Management Operations - Multi-file input and single
    file output. Simple structural changes, simple edits.
    If your module meet one of above, enter a number otherwise
    a 0.

kb_question(cplx4,
['Type of Module:
(13) Control Operations - Highly nested structure programming
    operators with many compound predicates. Queue and stack
    control. Considerable intermodule control.

(14) Computational operations - Basic numerical analysis:
    multivariate interpolation, ordinary differential equations.
    Basic truncation, roundoff concerns.

(15) Device-dependent Operations - Operations at physical I/O
    level (physical storage address translations; seeks, reads,
    etc.) Optimized I/O overlap.

(16) Data Management Operations - Special purpose subroutines
    activated by data stream contents. Complex data restructuring,
    at record level. If your module meet one of above, enter
    a number otherwise 0.

kb_question(cplx5,
['Type of Module:

(17) Control Operations - Reentrant and recursive coding.
    Fixed-priority interrupt handling.

(18) Computational operations - Difficult but structured
    N.A.: near singular matrix equations, partial differential
    equations.

(19) Device-dependent Operations - Routines for interrupt
    diagnosis, servicing, masking. Communication line handling.

(20) Data Management Operations - A generalized, parameter-
    driven file structuring routine. File building, command
    processing, search optimization. If your module meet one
    of above, enter a number otherwise a 0.

kb_question(cplx6,
['Type of Module:

(21) Control Operations - Multiple resource scheduling with
    dynamically changing priorities. Micro-code-level control.

(22) Computational operations - Difficult and unstructured
    N.A.: highly accurate analysis of noisy, stochastic data.

(23) Device-dependent Operations - Device timing-dependent
    coding, micro-programmed operations.

(24) Data Management Operations - Highly coupled, dynamic
    relational structures. Natural language data management.
    If your module meet one of above, enter a number otherwise
    a 0.

    item('Add','+').
    item('Subtract','-').
    item('Multiply','*').
    tem('Divide','/').

/* Intermediate COCOMO Software Development Effort Multipliers
   Table. Created Date: Mar 2, 1992 \hsize{1.0in} by Shouli Wang

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
/*  1. Product Attributes -- Required software reliability. */
    cost_rate(rely,very_low,0.75).
cost_rate(rely,low,0.88).
cost_rate(rely,nominal,1.00).
cost_rate(rely,high,1.15).
cost_rate(rely,very_high,1.40).
/*  2. Product Attributes -- Data base size. */
    cost_rate(data,low,0.94).
cost_rate(data,nominal,1.00).
cost_rate(data,high,1.08).
cost_rate(data,very_high,1.16).
/*  3. Computer Attributes -- Execution time constraint. */
    cost_rate(time,nominal,1.00).
cost_rate(time,high,1.11).
cost_rate(time,very_high,1.30).
cost_rate(time,extra_high,1.66).
/*  4. Computer Attributes -- Main storage constraint. */
    cost_rate(stor,nominal,1.00).
cost_rate(stor,high,1.06).
cost_rate(stor,very_high,1.21).
cost_rate(stor,extra_high,1.56).
/*  5. Computer Attributes -- Virtual machine volatility. */
    cost_rate(virt,low,0.87).
cost_rate(virt,nominal,1.00).
cost_rate(virt,high,1.15).
cost_rate(virt,very_high,1.30).
/*  6. Computer Attributes -- Computer turnaround time. */
    cost_rate(turn,low,0.87).
cost_rate(turn,nominal,1.00).
cost_rate(turn,high,1.07).
cost_rate(turn,very_high,1.15).
/*  7. Personnel Attributes -- Analyst capability. */
    cost_rate(acap,very_low,1.46).
cost_rate(acap,low,1.19).
cost_rate(acap,nominal,1.00).
cost_rate(acap,high,0.86).
cost_rate(acap,very_high,0.71).
/*  8. Personnel Attributes -- Application experience. */
cost_rate(aexp,very_low,1.29).
cost_rate(aexp,low,1.13).
cost_rate(aexp,nominal,1.00).
cost_rate(aexp,high,0.91).
cost_rate(aexp,very_high,0.82).

/* 9. Personnel Attributes -- Programmer capability. */

cost_rate(pcap,very_low,1.42).
cost_rate(pcap,low,1.17).
cost_rate(pcap,nominal,1.00).
cost_rate(pcap,high,0.86).
cost_rate(pcap,very_high,0.70).

/* 10. Personnel Attributes -- Virtual machine experience. */

cost_rate(vexp,very_low,1.21).
cost_rate(vexp,low,1.10).
cost_rate(vexp,nominal,1.00).
cost_rate(vexp,high,0.90).

/* 11. Personnel Attributes -- Programming language experience. */

cost_rate(lexp,very_low,1.14).
cost_rate(lexp,low,1.07).
cost_rate(lexp,nominal,1.00).
cost_rate(lexp,high,0.95).

/* 12. Project Attributes -- Use of modern programming practices. */

cost_rate(modp,very_low,1.24).
cost_rate(modp,low,1.10).
cost_rate(modp,nominal,1.00).
cost_rate(modp,high,0.91).
cost_rate(modp,very_high,0.82).

/* 13. Project Attributes -- Use of software tools. */

cost_rate(tool,very_low,1.24).
cost_rate(tool,low,1.10).
cost_rate(tool,nominal,1.00).
cost_rate(tool,high,0.91).
cost_rate(tool,very_high,0.83).

/* 14. Project Attributes -- Required development schedule. */

cost_rate(sced,very_low,1.23).
cost_rate(sced,low,1.08).
cost_rate(sced,nominal,1.00).
cost_rate(sced,high,1.04).
cost_rate(sced,very_high,1.10).
Appendix E

Expert SHELL1
This program is designed based on the program XSHELL, written by Covington, Nute, and Vellino[51]. An expert system consultation driver to be used with separately written knowledge bases. Procedures in the file include SHELL, SHELL_AUX, FINISH_SHELL, CALCULAT, DRIVER, MODULE, PARM, PARMSET, PARMRANGE, EXPLAIN, MEMBER, and WAIT. Requires various procedures defined in the files READSTR.PRO, READNUM.PRO, WRITELN.PRO, and YES.PRO.

/* shell1
As a query, this predicate begins a consultation. It is a procedure for the expert system consultation driver. It always succeeds. */

start :- kb_intro(Statement), writeln(Statement), nl, menu_select, end_pro.
end_pro :- !.

menu_select :-
write('-------------------------------------------'),nl,
write('1. Estimation of Software Product Cost according to COCOMO Mode (demo).'),nl,
write('2. Exit to MAIN MENU.'),nl,
write('Enter a number choice >>>'),nl,
read(Choice),nl,
proc(Choice).

proc(2) :- !,
   end_proc.
proc(1) :- !,
    put(1),nl,
    shell,
    menu_select.
end_proc :- !.

shell :
    assert(count_cplx(0)),
    kb_identify(ID),
    kb_identify_type,
    asserta(known(identification,ID)),
    writeln(ID), nl,
    distinguish_mode, nl,nl,
    write('Effort adjustment factor(product of effort multipliers
is following'),nl,
    nl,
    alter_cost_driver_rate,nl,nl,
    write('The resulting estimated effort for the project is
then'),nl,nl,
    kb_report(Phrase), nl,
    write(Phrase), nl,
    explain,
    kb_unique(no),
    !,
    shell_aux.

shell :- shell_aux.

/*
 * shell_aux
 * Prevents an abrupt end to a consultation that ends
 * without an identification, or a consultation where
 * multiple identifications are allowed.
 */

shell_aux :- not known(identification,_),
    abolish(known,2),
    write('You might give a wrong answer, please try again.....'),nl,
    write('You should start from beginning !!!!!!!!'),nl,
    !,
    finish_shell.

shell_aux :- kb_unique(no),
    known(identification,_),
    writeln('The all step have been finished, we will give a formal
report .'),
    wait,
    !,
    finish_shell.

shell_aux :- finish_shell.
finish_shell

* Erases the working database and asks if the user wants to conduct another consultation. Use retractall instead of abolish in some Prolog implementations.

finish_shell :-
a1tdri,
tell('expert_report1'),
listing(software_mode),
listing(mode_feature),
listing(cost_driver_rate),
listing(adjustment_factor),
told,
write('Some knownledge base have been put in a file called expert_report1!'),
wait,
abolish(known,2),
abolish(mode_feature,2),
abolish(software_mode,1),
abolish(cost_driver_rate,4),
abolish(adjustment_factor,1),
abolish(answers1,2),
write('If you think that this estimation does not meet your budget'),nl,
write('EXPERT will help you to continue work on the estimation for'),nl,
write('your project cost ....'),nl,
system('"mmi"'),
write('Do you want to conduct another consultation?'),nl,
yes('>'), nl, nl,
!,
shell.

finish_shell.

* To identify COCOMO Software Development Modes.

* /

calculat(Mode,_,Value1) :- known(Mode,StoreValue1), !,
Value1 = StoreValue1.

calculat(Mode,n,Value1) :- kb_question(Mode,Question), writelnn(Question),
write('>'),
readnumber(Responses1), nl,nl,
assert(mode(Responses1)),
assert(known(Mode,Responses1)),
To determine effort multipliers: each of the 15 cost driver attributes in COCOMO has a rating scale and a set of effort multipliers which indicate by how much the nominal effort estimate must be multiplied to account for the project's having to work at its rating level for the attribute. These cost driver attributes and their corresponding effort multipliers are asked questions in IAESP.Pro knowledge base, the results of applying these knowledge are given by the following drvier.

```
driver(Attribute,_,Value2) :- known(Attribute,StoredValue2),
                      !,
                      Value2 = StoredValue2.

driver(Attribute,n,Value2) :- kb_question(Attribute,Question),
                           writeln(Question),
                           readnumber(Response2), nl, nl,
                           assert(known(Attribute,Response2)),
                           assert(answer1(Attribute,Response2)),
                           !,
                           member(Response2,Value2).
```

```
module(Property)
```

```
module(Attribute,_,Value3) :- known(Attribute,StoredValue3),
                      !,
                      Value3 = StoredValue3.

module(Attribute,n,Value3) :- kb_question(Attribute,Question),
                           writeln(Question),
                           readnumber(Response3), nl, nl,
                           assert(known(Attribute,Response3)),
                           !,
                           member(Response3,Value3).
```

```
parm(Parameter,Type,Value)
```

```
* Type determines whether Value is to be a character, an atom, or a number. Value becomes the remembered value for the parameter if there is one. Otherwise the user is asked for a value and that value is remembered. When used as a test condition, Value is instantiated before the procedure is called and parm(Parameter,Type,Value) only succeeds if the remembered value, or alternatively the value reported by the user, matches Value.
* /
```
parm(Parameter,_,Value) :- known(Parameter,StoredValue), !, Value = StoredValue.

parm(Parameter,c,Value) :- kb_question(Parameter,Question), writeln( Question ), write( ' > ' ), read( Response ), nl, nl, assert( known(Parameter,Response) ), !, Value = Response.

parm(Parameter,a,Value) :- kb_question(Parameter,Question), writeln( Question ), readatom( Response ), nl, nl, assert( known(Parameter,Response) ), !, Value = Response.

parm(Parameter,n,Value) :- kb_question(Parameter,Question), writeln( Question ), readnumber( Response ), nl, nl, assert( known(Parameter,Response) ), !, Value = Response.

/*
parmset( Parameter, Type, Set )
* Type indicates whether the Parameter takes a character,
* an atom, or a number as value, and Set is a list of
* possible values for Parameter. A call to the procedure
* succeeds if a value for Parameter is established that is
* a member of Set.
*/

parmset( Parameter, Type, Set ) :- parm( Parameter, Type, Value ), member( Value, Set ).

/*
parmrange( Parameter, Minimum, Maximum )
* Parameter should be a parameter that takes numbers as
* values, and Minimum and Maximum should be numbers. A
* call to the procedure succeeds if a value for Parameter
* is established that is in the closed interval
* [Minimum,Maximum].
*/

parmrange( Parameter, Minimum, Maximum ) :-
parm( Parameter, n, Value ),
Minimum =< Value,
Maximum >= Value.
/*
 * explain
 * Explains how the expert system arrived at a conclusion
 * by finding an identification rule for the conclusion in
 * the knowledge base whose condition succeeds and showing
 * it to the user. If kb_explain(no) is in the knowledge
 * base explain merely waits for a keystroke to give the
 * user time to read the conclusion.
 */

explain :- kb_explain(yes), wait, !.

explain :- writeln(
 ['Do you want to see the rule that was used',
  'to reach the conclusion?'],
 not yes('>'), nl, !.

explain :- known(identification, ID),
 clause(kb_identify(ID), Condition),
 Condition, nl, nl,
 write('Rule: '),
 kb_report(Phrase),
 write(Phrase),
 writeln(ID),
 writeln(' if'),
 write(Condition), nl, nl, !.

/*
 * wait
 * Prints prompt and waits for keystroke.
 */

wait :- write('Press Return when ready to continue. '),
 get0(_), nl, nl.

member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).

/*
 * Procedure: Distinguish Software Development Mode.
 */

/*
 * distinguish_mode :-
 * mode(Value1),
 * modes(Value1,E_mode),
 * assert(software_mode(E_mode)),
 * write('Step1: Nominal Effort Estimation - PASS!'),nl,nl,
 * retract(mode(Value1)).
 */

/* The recursive multiplier function: calculating the rate of
development effort */
alter_cost_driver_rate :- alter_cost_driver_rate_aux,
write('Step2: Determine Effort Multipliers - PASS!'), nl, nl.

alter_cost_driver_rate_aux :- I is 1, Y is 1,
    assertz(answeryl(stop,stop)),
    alter_rate(I,Y),
    cost_rate(cplx,L,R),
    write('Cost Driver is cplx'), nl,
    write(' ', level is '), write(L), nl,
    write(' Rate is '), write(R), nl,
    adjustmment_factor(Product),
    Product_of_Multiplier is Product * R,
    number_mult(N),
    NewI is N + 1,
    abolish(number_mult,1),
    write(NewI), write(' Product is '),
    write(Product_of_Multiplier), nl,
    assertz(cost_driver_rate(cplx,L,R,Product_of_Multiplier)),
    write('Effort adjustment factor is'),
    abolish(adjustment_factor,1),
    assert(adjustment_factor(Product_of_Multiplier)),
    write(Product_of_Multiplier), nl.

alter_rate(I,Y) :- I > 14, !,
    assert(number_mult(I)),
    assert(adjustment_factor(Y)).

alter_rate(I,Y) :-
    answeryl(A,N),
    write('Cost Driver is '), write(A), nl,
    alter_rate(N,L),
    write(' ', level is '), write(L), nl,
    assertz(answeryl(A,N)),
    retract(answeryl(A,N)),
    cost_rate(A,L,R),
    write('Rate is '), write(R), nl,
    NewY is Y * R,
    write(I), write(' Product is '), write(NewY), nl,
    NewI is I + 1,
    assert(cost_driver_rate(A,L,R,NewY)),
    alter_rate(NewI,NewY).

altdri :- I is 1,
    altdri_aux(I).

altdri_aux(I) :- I > 8, !.

altdri_aux(I) :-
    known(X,Y),
    driver_feature(Y,Z),
    assert(mode_feature(X,Z)),
    retract(known(X,Y)),
    NewI is I + 1,
altdri_aux(NewI).
Appendix F

A Demo of IASCE Prototype
Script started on Wed Sep 2 13:18:34 1992
/home/soll/wmu/grad/swang/demo> rlogin s301
Last login: Tue Sep 1 23:27:30 from chip
SunOS Release 4.1 (Sun3-60) #1: Fri Apr 27 10:23:01 EDT 1990
The Computer Science lab will be closing for the semester on
Friday at 5 p.m. It will reopen in September.
/home/soll/wmu/grad/swang> cd demo
/home/soll/wmu/grad/swang/demo> cprolog
C-Prolog version 1.5
| ?- ['iasce.pro'].
readstr.pro consulted 1056 bytes 0.166667 sec.
readnum.pro consulted 1540 bytes 0.233333 sec.
writeln.pro consulted 180 bytes 0.0333337 sec.
yes.pro consulted 368 bytes 0.0500004 sec.
shell.pro consulted 8468 bytes 1.11667 sec.
menu.pro consulted 1236 bytes 0.15 sec.
mathmenu.pro consulted 1752 bytes 0.200001 sec.
kem.pro consulted 2564 bytes 0.466667 sec.
iasce.pro consulted 28652 bytes 3.81667 sec.

yes |
| ?- start.

DEMO for IAESP - An Itelligent Assistant for Estimation of
Software Product Cost Version 1.0

An Expert System for Estimation of Software Product Cost
To use the program, simply describe the software development mode
by answering the following questions.

1. Estimation of Software Product Cost according to COCOMO Mode (demo).
2. Exit to MAIN MENU.

Enter a number choice >>>>
| 1.

Step 1 - Nominal Effort Estimation: First, the following 8
questions are used to determine the project development mode.
There are three MODES
(1). ORGANIC (2). SEMIDETACHED (3). EMBEDDED
each mode has a corresponding scaling equation which is used to
determine the nominal development effort for the project in man-months as a function of the project size in KDSI (10000 delivered source instructions) and the project development mode. All these will be done by EXPERT SYSTEM... However, you need to answer the following questions; we assume you have a little bit project management experience. Good Lucky!

Organizational understanding of product objectives?
(t) Thorough
(c) Considerable
(g) General.
Enter a letter in lower case, followed by a period: >t.

Experience in working with related software systems?
(e) Extensive.
(c) Considerable.
(m) Moderate.
Enter a letter in lower case, followed by a period: >e.

Need for software conformance with per-established requirements?
(b) Basic.
(c) Considerable.
(f) Full.
Enter a letter in lower case, followed by a period: >b.

Need for software conformance with external interface specifications?
(b) Basic.
(c) Considerable.
(f) Full.
Enter a letter in lower case, followed by a period: >b.

Concurrent development of associated new hardware and operational procedures?
(s) Some.
(m) Moderate.
(e) Extensive.
Enter a letter in lower case, followed by a period: >s.

Need for innovative data processing architectures, algorithms?
(m) Minimal.
(s) Some.
(c) Considerable.
Enter a letter in lower case, followed by a period: >m.

Premium on early completion?
(l) Low.
(m) Medium.
(h) High.
Enter a letter in lower case, followed by a period:
>l.

Give your estimation of the Product size?
i.e. the size of the product as 10,000 delivered source
instructions(KDSI),
enter 10, which means 10 KDSI.
(1) less than 50 KDSI.
(2) less than 300 KDSI.
(3) All size.
Enter a number , followed by a period.
>>1.

Step2 - Determine Effort Multipliers; Each of the 15 cost driver
attributes in COCOMO has a rating scale and set of effort
multipliers which indicate by how much the nominal effort estimate
must be multiplied to account for the project having to work at its
rating level for the attribute. These cost driver attributes and
their corresponding effort multipliers are automatically given by
EXPERT SYSTEM, however you need answer the following question. We
assume you know a little project management, otherwise check the
manual of this EXPERT SYSTEM before you use it. Good Lucky!

Product Attributes - Question:
(RELY) Required software reliability ?
(1). very_low - Effect: slight inconvenience.
(2). low - Low, easily recoverable lossess.
(3). Nominal - Moderate, recoverable losses.
(5). very high- Risk to human life.
Enter a number.
>3

Product Attributes - Question:
(Data) Data base size ?
(2). low - (DB bytes)/(Prog. DSI )< 10.
(3). Nominal - 10 <= (DB bytes)/(Prog. DSI) < 100.
(4). Hgh - 100 <= (DB bytes)/(Prog. DSI) < 1000.
(5). very high- (DB bytes)/(Prog. DSI) >= 1000.
Enter a number.
>4

Computer Attributes - Question:
(time) Execution time constraint ?
(3). Nominal - <= 50% use of available execution time.
(4). High - <= 70% use of available execution time.
(5). Very high - <= 85% use of available execution time.
(6). Extra High - <= 95% use of available execution time.

Enter a number.
>5

Computer Attributes - Question:
  (stor) Main storage constraint ?
  (3). Nominal - <= 50% use of available storage.
  (4). High - <= 70% use of available storage.
  (5). Very high - <= 85% use of available storage.
  (6). Extra High - <= 95% use of available storage.

Enter a number.
>5

Computer Attributes - Question:
  (virt) virtual machine volatility ?
  virt means for a given software product, the underlying virtual
machine is the complex of hardware and software (Os, DBMS,etc.)
it calls on to accomplish its tasks.
  (2). Low - Major change every 12 months.
  (3). Nominal - Major: 6 months; Minor: 2 weeks.
  (4). High - Major: 2 months; Minor: 1 weeks.
  (5). Very high - Major: 2 weeks; Minor: 1 days.

Enter a number.
>4

Computer Attributes - Question:
  (turn) computer turnaround time ?
  (2). Low - Interactive.
  (3). Nominal - Average turnaround time< 4 hours.
  (4). High - 4 hours <= Average turnaround time<= 12 hours.
  (5). Very high - Average turnaround time> 12 hours.

Enter a number.
>4

Personnel Attributes - Question:
  (acap) Analyst capability ?
  Team rating criteria: analysis (programming) ability, efficiency,
ability to communicate and cooperate.
  (1). very_low - 15th percentile.
  (2). low - 35th percentile.
  (3). Nominal - 55th percentile.
  (4). High - 75th percentile.
  (5). very high- 90th percentile.

Enter a number.
>5

Personnel Attributes - Question:
(aexp) Application experience?
(1). very_low - <= 4 months experience.
(2). low - 1 year experience.
(3). Nominal - 3 years experience.
(4). High - 6 years experience.
(5). very high- 12 years experience.

Enter a number.
>4

Personnel Attributes - Question:
(pcap) programmer capability?
(1). very_low - 15th percentile.
(2). low - 35th percentile.
(3). Nominal - 55th percentile.
(4). High - 75th percentile.
(5). very high- 90th percentile.

Enter a number.
>4

Personnel Attributes - Question:
(vexp) Virtual machine experience?
(1). very_low - <= 1 month experience.
(2). low - 4 months experience.
(3). Nominal - 1 years experience.
(4). High - 3 years experience.

Enter a number.
>3

Personnel Attributes - Question:
(lexp) Programming language experience?
(1). very_low - <= 1 month experience.
(2). low - 4 months experience.
(3). Nominal - 1 years experience.
(4). High - 3 years experience.

Enter a number.
>3

Project Attributes - Question:
(modp) Use of modern programming practices?
(1). very_low - No use.
(2). low - Beginning use.
(3). Nominal - Some use.
(4). High - General use.
(5). very high- Routine use.

Enter a number.
>4
Project Attributes - Question:
(tool) Use of software tools?
(1). very_low - Basic microprocessor tools.
(2). low - Basic mini tools.
(3). Nominal - Basic midi/maxi tools.
(4). High - Strong maxi programming test tools.
(5). very high- Add requirements, design, management, documentation tools.

Enter a number.
>4

Project Attributes - Question:
(secd) Required development schedule?
(1). very_low - 75% of nominal.
(2). low - 85% of nominal.
(3). Nominal - 100% of nominal.
(4). High - 130% of nominal.
(5). very high- 160% of nominal.

Enter a number.
>4

Type of Module:
(1). Control Operations - Straightline code with a few nonnested structured programming operators: DOs, CASEs, IFTHENELSEs, Simple predicates.
(2). Computational operations - Evaluation of simple expressions: e.g.,
   \[ A = B + C \times (D - E) \].
(3). Device-dependent Operations - Simple read, write statements with simple formats.
(4). Data Management Operations - Simple arrays in main memory.
   If your module meet one of above, enter a number otherwise a 0.
   >5

Type of Module:
(5). Control Operations - Straightforward nesting of structured programming programming operators: Mostly simple predicates.
(6).Computational operations - Evaluation of moderate-level expressions: e.g.,
   \[ D = SQRT(B**2-4.*,A*C) \].
(7). Device-dependent Operations - No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap.
(8). Data Management Operations - Single file subsetting with no data structure; changes, no edits, no intermediate are files.

If your module meet one of above, enter a number otherwise a 0.
>5

Step 1 - Nominal Effort Estimation has been done!
COCOMO Software Development is ORGANIC MODE ....
i.e. Batch data reduction, scientific models, business models
familiar OS, compiler, simple inventory, production control

Step 2 - Determine Effort Multipliers has been done!
Step 3 - Estimate Development Effort: We compute the estimated
development effort for a project as the Nominal development
effort times the product of the effort multipliers for the
15 cost driver attributes. It is starting now....

Enter estimation of KDSI:>10

$3.2 \times 10^{1.05} = 35.9045$ man-months

The nominal development effort for this project = 36 man-months

Step 1: Nominal Effort Estimation - PASS!

Effort adjustment factor (product of effort multipliers is following

Cost Driver is rely
  level is nominal
  Rate is 1
1  Product is 1

Cost Driver is data
  level is high
  Rate is 1.08
2  Product is 1.08

Cost Driver is time
  level is very_high
  Rate is 1.3
3  Product is 1.404

Cost Driver is stor
  level is very_high
  Rate is 1.21
4  Product is 1.69884

Cost Driver is virt
  level is high
  Rate is 1.15
5  Product is 1.95366

Cost Driver is turn
  level is high
  Rate is 1.07
6  Product is 2.09042

Cost Driver is acap
  level is very_high
  Rate is 0.71
7  Product is 1.48419

Cost Driver is aexp
  level is high
  Rate is 0.91
8  Product is 1.35062

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Cost Driver is pcap
  level is high
Rate is 0.86
9  Product is 1.16153
Cost Driver is vexp
  level is nominal
Rate is 1
10 Product is 1.16153
Cost Driver is lexp
  level is nominal
Rate is 1
11 Product is 1.16153
Cost Driver is modp
  level is high
Rate is 0.91
12 Product is 1.05699
Cost Driver is tool
  level is high
Rate is 0.91
13 Product is 0.961861
Cost Driver is sced
  level is high
Rate is 1.04
14 Product is 1.00033
Effort adjustment factor is 1.00033
Step2: Determine Effort Multipliers - PASS!

Press Return when ready to continue.

The all step have been finished, we will give a formal report.
Press Return when ready to continue.

Some knowledge base have been put in a file called expert_report1!Press Return when ready to continue.

If you think that this estimation does not meet your budget
EXPERT will help you to continue work on the estimation for
your project cost ....

An report derived from Expert System ----------------->

=================================================================
The following is an report of estimation of your project cost
=================================================================

<table>
<thead>
<tr>
<th>Software Mode</th>
<th>Nominal Cost(MM)</th>
<th>Cost(MM)</th>
<th>dollar</th>
</tr>
</thead>
<tbody>
<tr>
<td>organic</td>
<td>35.9045</td>
<td>30.529</td>
<td>$155000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost Drivers</th>
<th>Ratings(level)</th>
<th>Ratings</th>
<th>Effort Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature</td>
<td>level</td>
<td>factor</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>rely</td>
<td>nominal</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>high</td>
<td>1.08</td>
<td></td>
</tr>
<tr>
<td>cplx</td>
<td>low</td>
<td>0.85</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>very_high</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>stor</td>
<td>very_high</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>virt</td>
<td>high</td>
<td>1.15</td>
<td></td>
</tr>
<tr>
<td>turn</td>
<td>high</td>
<td>1.07</td>
<td></td>
</tr>
<tr>
<td>acap</td>
<td>very_high</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>aexp</td>
<td>high</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>pcap</td>
<td>high</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td>vexp</td>
<td>nominal</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>lexp</td>
<td>nominal</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>modp</td>
<td>high</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>tool</td>
<td>high</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>sced</td>
<td>high</td>
<td>1.04</td>
<td></td>
</tr>
</tbody>
</table>

Effort adjustment factor (product of effort multiplier) = 0.850284

Do you want to conduct another consultation?  
\( > n \)

1. Estimation of Software Product Cost according to COCOMO Mode (demo).  
2. Exit to MAIN MENU.

Enter a number choice >>>>  
\( : 2 \).

yes  
\( ? - \) halt.

[ Prolog execution halted ]
/home/sol1/wmu/grad/swang/demo> exit

script done on Wed Sep 2 13:22:19 1992
REFERENCES


