MODELLING OF SOFTWARE DEVELOPMENT EFFORT FOR THE FOURTH GENERATION ENVIRONMENT

by

VIJAY K. KANABAR

A Thesis
Presented to the University of Manitoba
in Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Interdisciplinary
Department of Actuarial and Management Sciences
and Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

ISBN 0-315-77925-X

MODELLING OF SOFTWARE DEVELOPMENT EFFORT FOR THE FOURTH GENERATION ENVIRONMENT

BY

VIJAY K. KANABAR

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

© 1992

ABSTRACT

This thesis deals with the topic of estimating software development effort when *fourth generation tools* (4GTs) such as form and report generators are used. Traditional predictors or cost models are inadequate for measuring and estimating development effort involving 4GTs. Such traditional predictors or models are more oriented towards measuring "manual coding" than towards "specification-oriented coding". An innovative predictor called *specification element* (SE), embedded in a *4GT Model*, is introduced here to measure application development effort using 4GTs. SEs are associated with data and screen field elements for purposes of effort estimation. Knowledge-based techniques are used to refine effort estimates provided by the 4GT Model for the influence of *project factors* such as "developer experience" and "familiarity with tools".

To my mother Chandrika Kanabar and

my father Kalyandas Kanabar

Acknowledgements

Completing a Ph.D. degree is a lengthy and demanding process. Several people were directly involved and all contributed a major chunk of their valuable time. To begin with, I would like to acknowledge the contribution of Dr. Eric Seah who functioned as my advisor. He provided the necessary intellectual freedom all along and was readily available even on weekends for consultation and for reading and commenting on my thesis chapters. He readily provided me financial support (often without asking!) from his NSERC and related grants to attend conferences and present papers. Dr David Scuse very effectively contributed throughout my graduate programme. His insights into my research proved to be very valuable; he provided "expert" advise with regards to thesis directions and rules & regulations as well. Dr Edwin Cheng, a recent addition to the committee, proved to be very helpful — he had to quickly master a lot of my research and edit the thesis.
Dr Tarek Abdel-Hamid's involvement and thesis evaluation significantly improved the quality of the final product. Dr Ken Hughes, Dean of Graduate Studies, did a great job as well; he wanted me to complete a truly first-class and useful Ph.D. programme. Rita Campbell put her heart and soul in type-setting this thesis, many thanks for a job well done. I would also like to thank the various members at Great-West Life, Investors Syndicate, and the U of W for providing useful data/comments.

Finally, I would like to acknowledge the contribution of my wife Dina Kanabar, more than me, she has earned the Ph.D.— if not the Ph.T. (Pushing Hubby Through). Without much of a choice she spent several lonely days and nights raising our young ones Meera and Anish.

Table of Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

1.1 Statement of the Problem

This thesis presents a model for measuring and estimating effort of software applications developed by end-users or programmers with tools broadly referred to as *Fourth Generation Languages* (4GLs).

The importance of estimating the size and time required for software development cannot be over-emphasized. Consider the following quotation by Barry Boehm in his foreword to Tom DeMarco's book:[1]

> Better cost estimation methods help us to understand the relative costs and benefits of a proposed future system well enough to be able to reduce its scope or to eliminate portions whose benefits do not justify their estimated costs.

Within the realm of 4GLs, small to medium business applications developed using tools such as query facilitators, form generators, report generators, application generators, graphics languages, and specification oriented application packages, are the focus of this thesis. Such software tools facilitate application creation without conventional programming as we know it, and are referred to as *fourth generation tools* (4GTs) in this thesis. According to Pressman:[6]

> The term fourth generation techniques (4GT) encompasses a broad array of tools that have one thing in common: each enables the software developer to specify some characteristic of software at a high level. The tool then automatically generates source code based on the developer's specification. There is little debate that the higher the level at which the software can be specified to a machine, the faster a program can be built. (p. 24)

In this thesis we use the term "Fourth Generation Tools" (4GTs) as opposed to the term "Fourth Generation Languages" (4GLs) as it is the tools, rather than languages, that our research focuses on. Other researchers also commonly use the term "Fourth Generation Tools" for the same reasons.[98]

The importance of 4GTs is evident from the following summary by Roger Pressman:[6]

> To summarize, fourth generation techniques are likely to become an increasingly important part of software development during the next decade. As the figure (reproduced as Figure 1.1 below) illustrates, the demand for the software will continue to escalate throughout the remainder of this century, but conventional methods and paradigms are likely to contribute less and less to all software developed (p. 25).

For simplicity, the model at present does not consider larger applications such as those exceeding thousand person-days within its scope. Such applications usually require extensive 3GL procedural coding, and end-users will not be typically involved with such projects.

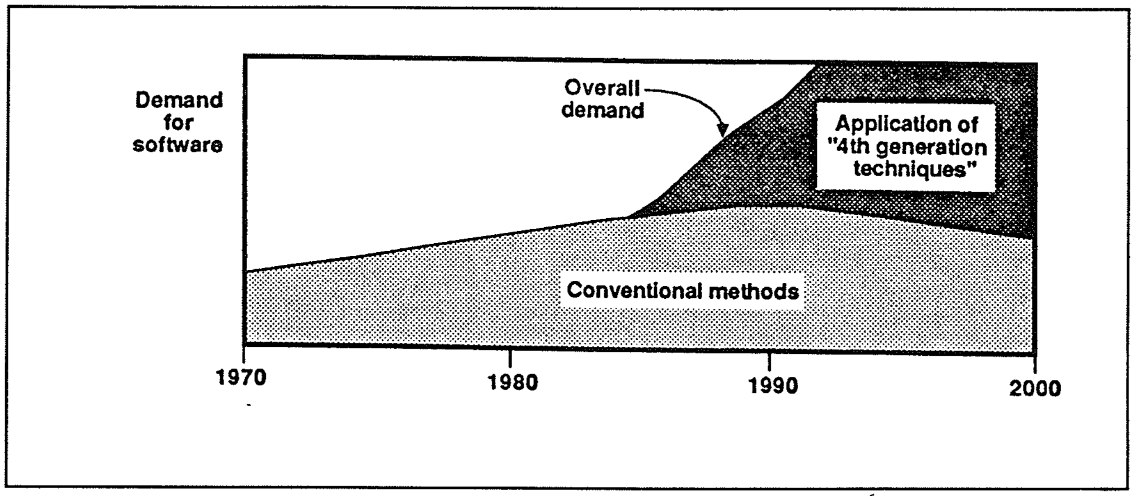


Figure 1.1: Changing Nature of Software — Pressman[6]

1.2 Issues and Objectives

This section describes the problems of estimating size and effort in the fourth generation environment. The issues related to 4GTs are examined first, followed by prototyping issues. There are several reasons for cost modelling 4GT-based application development. To begin with, the problems in this domain are unique:

- Due to the significant improvements that have taken place in software and hardware technology, powerful fourth generation tools and techniques for developing applications have emerged in the market place. When compared with third generation languages these tools have successfully resolved some of the problems of traditional software development problems, such as poor quality, high cost of development and maintenance, and very slow development rate.

- Several firms have indeed become productive today due to adoption of this new software technology. James Martin, citing several cases, indicates that these gains represent the largest step forward in application creation since the invention of programming.[7] This may be attributed to the fact that overall development and programming effort is significantly reduced due to simplicity in computing.
- As a solution to decreasing the huge application backlog, end-user computing using 4GLs is being advocated by managers and M.I.S. professionals. Microcomputer based database management systems have made it feasible for end-users to create forms, reports, and even complete applications easily.[8,9] However, there are many risks associated with such a strategy, especially delayed schedules and unanticipated costs.
- Literature survey reveals that estimating systems development effort in the fourth generation domain is difficult.[10] Moreover, the existing predictors and models are not suitable for estimating application development effort using 4GTs — such models are more directed

towards estimating code-oriented programming languages rather than specification-oriented applications.

The above facts indicate that suitable models for estimating 4GT-based application should be researched. Also, since 4GTs are closely associated with different development paradigms, such as *evolutionary prototyping* or *throw-away prototyping*, any such cost model must be flexible enough to accommodate the impact of such techniques as well.

1.3 Thesis Overview

In Chapter 2, we review the literature and describe the major approaches to project planning and cost estimation taken by researchers. It also introduces topics such as application prototyping, software metrics, project management, and knowledge-based systems. Chapter 3 introduces the *4GT Model* for estimating application development effort when 4GTs are used. Chapter 4 describes *PFES*, a knowledge-based system for evaluating project factors. 4GT model experimentation, analysis, and validation are presented in Chapter 5. Finally, Chapter 6 presents conclusions and topics for further research. The basic 4GT Model is summarized next.

1.3.1 Modelling 4GT-Based Applications

In the fourth generation environment, application development effort is dependent on the total number and size of functions to be implemented. A function can be classified into one of the following function types: form, report, process, and data.

In order to estimate the effort involved in implementing each function we use a new predictor called *specification element* (SE). This term is a hybrid of the two terms *specification,* and *screen-, data-element* and is therefore defined as, "a specification task associated with implementing a data or screen field element". SEs represent information system size and are coupled with screen fields or data elements by design.

Each SE has an effort value in person-hours associated with it. This value represents the work effort required to implement one SE and hence one screen field. By considering all the screen fields and data elements of an application and their relative specification effort, we can obtain the total development effort.

Finally, as the 4GT Model only estimates average development effort, *project factors* such as "familiarity with tools", and "programmer experience" are evaluated separately, to produce a refined effort estimate.

1.3.2 Conclusions

The contributions of the thesis are in two areas:

- The first and most significant contribution of this thesis is the development of the 4GT Model for estimating application development effort when specification-oriented tools are used. At present no such model exists. The model is capable of estimating in two modes — *ball-park* and *base-line* — the former provides rough estimates early on in the development cycle, whereas the later provides more accurate estimates but only after some design is complete. In this regard we also classify different 4GT development paradigms and determine their impact on the effort estimation model.
- The second contribution is the knowledge-based framework for evaluating *project factors* (PFs) The notable feature here is the ability of the model to take into consideration application development by end-users. The knowledge-based approach provides an opportunity to use "what if" analysis to investigate the impact of various project parameters.

Chapter 2

Review

2.1 Introduction

Software modelling for cost estimation and project management has been a major research issue for both researchers and practitioners for many years. Cost models have been successfully developed for the software industry, especially for the third generation software development domain. Some of these models are able to give a good cost estimate if calibrated properly to the users environment. As a result project managers can use such models for planning and estimating software development in addition to relying on their experience for making decisions.

Recently powerful fourth generation tools and techniques for developing software have emerged in the market place. When compared with third generation languages, they have successfully solved some of the problems of traditional software development, such as poor quality, high cost of development and maintenance, and very slow development rate. This chapter reviews fourth generation tools and techniques, and traces the progress of current state of art in cost modelling. It also describes the relationship of the proposed model to previous research.

2.2 Fourth Generation Languages, Tools, and Techniques

Various concepts associated with modelling such as project management, software metrics, software development process, application prototyping, fourth generation technology, and knowledge-based approach, are introduced in this section.

2.2.1 Project Management

Project Management in the context of software development is discussed here. The fundamental concepts, however, are quite similar to those in other industries such as construction or engineering. Managing a project typically involves using a project management methodology. This generally consists of the following phases: planning, scheduling, and controlling.

During the *planning* stage the project is broken down into smaller but more manageable components called tasks or activities. Work effort is estimated for each of these activities using past experience or historical data as a guide. A cost model, as discussed in this thesis, can play a useful role here for providing effort estimates. Once the work effort is estimated, a network can be created to show the sequence of activities that make up the entire project. In the *scheduling* stage we map the activities to a calendar, and determine start and finish dates for each task. Several project management tools can be used at this stage. They range from powerful mainframe based products such as IBM's Application System, to relatively smaller project management tools based on microcomputers such as Microsoft Project. Such tools contribute significantly to project scheduling by providing various graphs, including those that identify the critical path (CPM/PERT). The final stage, *control*, ensures that the entire project is completed on time and within budget. Adequate control ensures that the end products are of good quality, within budget, and on schedule.

2.2.2 Software Metrics

Software metrics are quantitative measures of various characteristics of projects. They measure code and documentation, development process, development activities, the problem domain, and environment characteristics like people, tools, or techniques used.[12]

Many firms today have adopted a software metrics approach, to assist them with various aspects of software development. Such a strategy implies that databases containing various development related metrics such as cost-oriented data and size-oriented data are accumulated for all software development projects. While the project is in progress, and especially on completion, various productivity and quality metrics are generated from such a database. This information is used to improve the quality of future projects and to evaluate the impact of new tools and techniques. Grady and Caswell have described Hewlett-Packard's metrics program in detail in their recent book.[13] They indicate that HP had two objectives in mind when they initiated their program — first, an improvement in productivity; and second, an ability to measure tools (developed in-house or purchased) for effectiveness. With regards to the first point, they felt that the very act of measuring the software development process itself would lead to short-term improvements in productivity. They quote Peters and Waterman's classic book *In Search of Excellence* to illustrate their point:

> People ... like to perform against standards — if the standard is achievable, and especially if it is one they played a role in setting.

Several hundred people were involved in HP's software metrics program. And after three years of commitment they were able to achieve several advantages; the most important of these (from the CASE* perspective and organizational perspective as well) are:

- Ability to measure progress.
- Ability to identify practices which lead to the highest quality and productivity.
- Ability to estimate and schedule projects better.

* CASE stands for *computer assisted software engineering* — it refers to the application of automated technologies to software engineering procedures.

When establishing a software metrics program, Grady and Caswell recommend that the following key steps be performed: assign software metrics responsibility to specific people; convince people of the importance of these metrics and indicate that accuracy depends upon their willingness to take the time to collect data; define metrics to be collected (such as size, defects, effort, and cost); try to automate data collection; and, create a metrics database.

One of the most significant advantages of the software metrics approach is its ability to assist us with project estimation. According to Pressman all estimation techniques use software metrics (p.43):[6] "Software metrics (past measurements) are used as a basis from which estimates are made".

2.2.3 Gathering Project Metrics

The project manager's emphasis is typically on completing the project on schedule. Yourdon[14] states, the typical problem with documenting software metrics is that most managers do not get enthusiastic about investing 5% of the project team's resources *this year*, in order to provide some data that will be useful to some other project manager *next year*. The end result is that project planning and predicting effort is difficult each time around. Therefore, a certain degree of project planning automation should benefit an organization. In order to overcome the problems with gathering project metrics, mechanization of the metrics gathering process is desirable, and Yourdon recommends the use of automated tools to assist us with all the stages of measurement. CASE tools can play a very useful role here as they are highly integrated and automated.

2.2.4 Software Development Process

Several life cycle models have been used by organizations to assist them with software development. The models are phase oriented and use distinct stages:

- requirements specification
- design
- implementation
- testing
- installation

In the life cycle model above, if a rigorous approach (traditional, non-prototyping) is used then the software product materializes only after implementation. But if the final product does not reflect user requirements adequately, or is imperfect in any other way, then the development can prove to be very expensive since some of the details have to be worked over again. In order to overcome this problem application prototyping is often used as a development strategy. Even though there are several techniques for prototyping, they all have one common objective — demonstration of a working system to the users, as early as possible. Table 2.1 describes some of these techniques. While all of the illustrated strategies assist us with eliciting user requirements early on in the system development life cycle, some are more risky than the others. For instance, screen/simple mock prototyping is considered more risky because its functionality is not fully tested up front.

Table 2.1: Prototyping Techniques

Throwaway	Prototype is discarded after acceptance by the users.
Rapid / Evolutionary	Prototype is used and refined until it becomes the final product.
Detailed / Full	Throwaway prototype that mimics the functionality of the final application completely.
Screen / Simple mock up	Prototype mimics the screen layout of the application.

2.2.5 Fourth Generation Languages

This refers to a set of tools and languages primarily associated with a database management system. The tools today consist of the following components: data dictionary, software generators, fourth generation languages, report writers, screen generators, spreadsheet, presentations graphics, and query languages. Fourth generation languages and tools have reduced the time and effort required to generate an application by a factor of at least 5 to 10, when compared with application development using 3GL's; this is possible largely because general applications are built using high-level specifications.[15] In conjunction with techniques such as prototyping, fourth generation tools have succeeded in improving the productivity of software developers, and have enhanced the quality of software being designed. Here overall programming effort is significantly reduced due to *simplicity in computing*. This can be attributed to integrated database systems, form-, and report-generators, non-procedural 4GLs, query languages, and controlled use of alien syntax and mnemonics — allowing the developer to concentrate on software development.

Figure 2.1 describes the key components of the fourth generation architecture tools and their links with the DBMS. It illustrates the central role of the DBMS in the environment. The forms generator, the report writer, the application generator, and the query language play an important role in developing applications rapidly. These tools are described next.

Data Dictionary: It serves as a central reservoir of all data. It contains information on files, usage of data, metadata. In the *CASE* approach the data dictionary plays a key role in applications development.

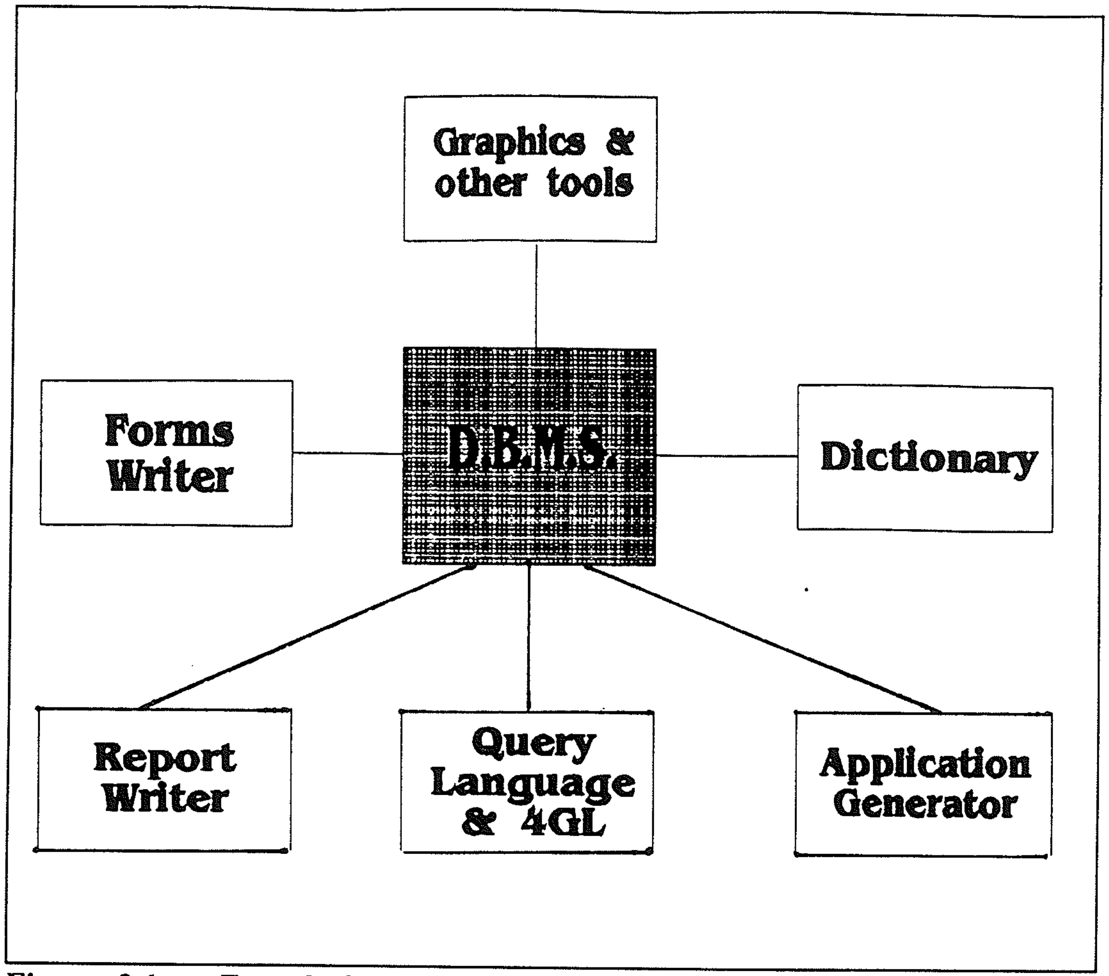


Figure 2.1: Fourth Generation Software Components

Query Language: It is a non-procedural language and is used to communicate with the database. It consists of data definition statements that facilitate the creation and description of a database, data manipulation statements that deal with retrieval and update of the database, and data control statements that specify security constraints. Structured Query Language (SQL) is the undisputed query language and industry standard today for all relational databases.

Forms Generator: This is a flexible interactive facility used to create forms for data entry, query, update, and deletion. Forms today have a rich set of features that permit one to generate many types of complex applications. For instance, forms permit us to perform data validation, restrict access, generate sequence numbers for primary keys, table lookups, create triggers, and use computed values. Such functionality until recently had to be hard coded using a programming language.

Report Generator: This is also an interactive tool for creating reports. Report generators today are very powerful and are capable of producing virtually any type of output format. However, it must be added here that reports can also be created using non-procedural query language statements.

Program Generator: There are a wide variety of program generators available today. They range from simple menu and module/procedure generators to full-fledged application generators. The common thread here is that very little physical coding is actually done — specifications serve to provide instructions to the program generator. The above products are integrated today using a common query language. For instance, ORACLE's 4GL tools such as the SQL*FORM, and SQL*REPORTWRITER support the creation of reports and forms using its query language SQL.

2.3 Software Models for Estimation and Management

Estimating the cost of a project is probably the most tedious task for a planner and a lot is at stake. Planners usually rely on historical data and accumulated experience to develop estimates. But it is not uncommon to find organizations very lax when it comes to recording project data. A novice estimator with little experience and no access to any historical data would therefore suffer the most under such circumstances. It is worthwhile for such estimators to have access to a software costing model.

On reviewing the literature three types of representative software models for cost estimation and project planning have been identified. Each differs from the other distinctly, but they collectively serve to describe the options available to planners when selecting tools for software cost estimation and management. The three representative models are:

(i) COCOMO

(ii) Function Point Analysis

(iii) System Dynamics Model

2.3.1 COCOMO

COnstructive COst MOdel (COCOMO) introduced by Boehm[16] consists of three increasingly complex models — *basic, intermediate,* and *detailed.* Three modes of software development exist for the above models (see Table 2.2).

Table 2.2: COCOMO Development Mode

Organic	A small team of experienced programmers develop software in a familiar environment.
Semidetached	The composition of programmers is a mix of novice and experienced, and the environment not totally familiar.
Embedded	The project has tight constraints and the problem is unique; past experience may not help a lot.

The *basic* model lacks accuracy since it does not consider the variable project parameters of the software development environment. This situation is addressed in the *intermediate* model which introduces a set of 15 cost drivers, such as product complexity, analyst capability, and programming language experience. The *detailed* model provides two additional features — phase sensitive effort multipliers for each cost driver, and a three-level product hierarchy (module, subsystem, and system levels) for rating the cost driver.

Even though the COCOMO model is easy to use, commercially available, and well documented, it has some shortcomings. To begin with, it uses *lines of code* (LOC) as a predictor — this is a low level metric, and it requires us to estimate the LOCs for the new application very early on in the system development life cycle. Estimating the LOCs required for the new application before proper analysis or design is done can result in a loss of accuracy. Another problem with COCOMO is that it places the onus on the estimator to select the correct type of model and development mode.

The COCOMO model is based on data from 3GL projects measured at TRW; hence the nature of the projects involved in their bench mark is "manual coding" oriented. Their model is clearly not relevant to the development of fourth generation applications — especially to those that are developed using "specification oriented" techniques.

2.3.2 Function Point Analysis

Albrecht's *Function Point Analysis* (FPA) model[18] was introduced more than a decade ago in the context of productivity measurement. FPA is independent of any language and it estimates the size of an application on the basis of the number of inputs, outputs, files, interfaces, and inquiries (see Table 2.3). Once the requirements have been defined one can use FPA to identify function points and

Table 2.3: Function Point Classification

Outputs	Application oriented information processed by the computer for the user. This includes entire reports, and screens but excludes individual data items within a report or a screen.
Inputs	Application oriented information entered by the user for the computer to process. Includes updates, ie, add, modify, or delete.
Inquiries	These are queries to the database for information. No updates are performed.
Files	Logical files used by the application.
Interfaces	Refers to external files interacting with the existing application.

classify them into one of five categories (see Table 2.3). At the same time the complexity of the function must also be identified as simple, average, or complex.

Finally, fourteen overall-adjustment factors are used to make adjustments for system characteristics to provide the *total correction value.* The final function point count is obtained by using the following equation:

Final FP = Total Unadjusted Fps * [0.65 + (.01 * Total Correction Value)]

Largely due to its ability to use information available very early on in the project, the FPA has made some in-roads into the data processing industry for cost estimating business applications.

There are, however, several limitations with the Function Point Analysis approach. To begin with even though it is more suitable than COCOMO for estimating non-procedural language, it is incapable of cost estimating 4GTs. For instance, the definition for input and output (see Table 2.3) states that the individual data items within a report or a screen are to be ignored. Unfortunately, with 4GTs the fields act as a *cost centre* for accumulation of work effort and cannot be ignored. The 4GT Model deals with this issue elegantly.

A second major limitation with the Function Point Analysis method is that it has no avenue to address 4GT based development (such as using forms generators, report generators, or application generators). As stated by Dredger,[19] "if generators or report writers are used it is impossible to count function points for all possibilities, and you must settle the issue by crediting the application with one complex output." This is a major stumbling block for FPA. It does not perform well in a highly automated application development environment.

A major flaw with the FPA is that its weighting is inadequate when specification oriented programming occurs. (Here the primary development effort is based around screen fields. This aspect is described in detail in the next chapter.) For example, Symons questions why "a system component containing, say, over 100 data elements is given at most twice the points of a component with only one data element". We therefore regard FPA's weighting scheme as biased against 4GT based application development.

2.3.3 System Dynamics Model

The *system dynamics model* introduced by Abdel-Hamid[20,21] looks at the software estimation and planning issues within a much broader research program. Its objective is to comprehend, and to make predictions about the dynamics of the entire software development process. The model consists of four major subsystems which are described in Table 2.4.

Table 2.4: Subsystems of the Systems Dynamics Model

Human Resource Management	Deals with hiring, training, assimilation and transfer of human resources.
Software Production	Models activities such as designing, coding, and testing of software.
Controlling	Measures progress, perceived productivity, and determines effort still needed.
Planning	Plans work force and schedule.

The system dynamics model permits inexpensive simulation and controlled experimentation of different project management decisions, such as addressing the problems related to a project running behind schedule. The model has proven to be a useful tool for the study of software cost estimation.[22,23]

2.4 4GL-based Cost Estimation Research

This section reviews other 4GL-based cost estimation research. Verner and Tate have estimated size and effort for a project using a fourth generation application development system called ALL.[24] The technique they used is illustrated in Figure 2.2. As evident from the figure, they did not use any new model for 4GL effort estimation. Instead, they derived effort estimates by combining both the COCOMO and FPA models. Such a strategy must be applied carefully for several reasons. The conversion ratios assumed here such as "1 FP = 17 ALL" have not been validated using a large sample size. The strategy to use "1 FP = 110 COBOL" is also a controversial one since there can be a large variance in this ratio. Consider the following excerpt from Dreger describing the problems with interfacing models such as COCOMO with FPA (pages 132-33):[25]

> The most common way in which Function Point Analysis has been *misused* is using it to try to estimate source lines of code, from which the forecast, evaluation, or analysis is *then made*. This misuse of FPA is just plain WRONG! If industry cannot even decide to measure one line of code once it is written (one researcher found a huge 2300% variance in productivity due only to extremely wide variations in 7 SLOC definitions!), how can FPA possibly predict it before it is written? Moreover, this two-step process introduces two levels of error into the solution, the more serious of which is the plus-minus 50% (on average) distortion introduced when attempting to predict the number of COBOL source lines.

Use Function Point Analysis technique
to obtain total FP count
863 FP's

|
|

Use Language Expansion Ratio
(1 FP = 110 COBOL LOC)
LOC = 863 * 110 = 94930

|
|

Reduce the estimated LOC since a 4GL is being used
(Assuming 70% non-procedural content
the LOC is contracted by 16%)
new LOC = 94930 * .16 = 15188

|
|

Cross check effort with Jones Factors
(1 FP = 17 ALL code)
LOC = 863 FP * 17 = 14671

|
|

Use COCOMO to obtain effort & schedule
(15,000 LOC used as input parameter)

Figure 2.2: Verner-Tate Strategy for 4GL Effort Estimation

Integrating models such as COCOMO with FPA might therefore not be a very suitable strategy for 4GL effort-estimation. It is preferable instead to design an independent model for estimating 4GL projects.

In another related attempt, Wrigley and Dexter are researching the results of several FOCUS programs to see if they can come up with a reliable predictor of system size in terms of lines of code using reverse engineering techniques.[26] (Their research is largely oriented towards the examination of 4GLs not 4GTs.) They present a research model that establishes linkages among units of system requirements specification, design, and source code.

Their pilot study is unfinished — while they establish a link between information system size and LOCs, they do not extend their model to predict effort. They also do not address the issue of specification oriented programming. Nevertheless as their research is based in the fourth generation environment, any results obtained here are quite relevant to our thesis as well.

2.5 Knowledge-Based Systems

In this section we review the technology, as well as planning and cost estimation literature, in knowledge-based systems. Also the relationship of the thesis problem with other knowledge-based systems research is described here.

2.5.1 Knowledge-Based Systems Technology

Knowledge-based systems are application systems where domain knowledge is explicit and separate from rest of the system. Domain knowledge refers to all entities, facts and knowledge related to the application. Expert systems are specialized computer programs that use expert knowledge to attain high levels of performance in a narrow problem area. They mimic the reasoning of experts and are useful for very specific tasks such as medical diagnosis and computer configuration. Waterman[27] classifies expert systems as a subset of knowledge-based systems.

The domain knowledge is contained in the knowledge-base, that is, all facts and information pertaining to the application are represented in the knowledge-base. In contrast with conventional systems the data and knowledge are explicit and easily accessible. The architecture of an expert system consists of a *knowledge-base* and an *inference engine*. The knowledge base is further subdivided into two components *facts* and *rules*. Facts are known data about the system. Examples of facts are default values assigned to variables, eg., age of a person, or the date of joining. Some facts change from one query session to another, while others do not.

A rule is a formal way of specifying a resolution or indicating a decision. It is usually expressed as: "IF premise THEN conclusion", or "IF condition THEN action". The rule is the most common form of representation in a knowledge-based system. However, other structured models for representation exist.

The inference engine consists of programs that provide a general purpose problem solving mechanism for all queries. This is normally a backward chaining or a forward chaining mechanism that traverses across the knowledge base, executes rules and recommends solutions. If several rules are triggered concurrently, they are placed in a conflict set and resolved by the inference engine.

2.5.2 Knowledge-based Systems Research in Literature

Even though little attention has been paid to applying the technology directly in the areas of fourth generation based software cost estimation, or strategic and tactical planning, there is a continuing interest in using the knowledge-based approach for several aspects of project management. While it is beyond the scope of this thesis to review all aspects of project planning and scheduling applications using artificial intelligence (largely due to the abundance of such literature), an overview of artificial intelligence as it pertains to planning and cost estimation is presented below .

2.5.3 PAINTER: An Expert System for Cost Estimating

Biegel *et al.*[28] were interested in building a very general cost estimating shell that attempts to accommodate a wide variety of cost estimating situations with each having a specialized knowledge-base. Accordingly, *painter,* a rule-based cost estimating program for house-painting was designed using C language to run on the IBM PC microcomputer. The input data activate the appropriate decision table for evaluation. For instance when *painter* asks the user for the surface type to be painted — brick, stucco, wood, etc., each individual task calls the next appropriate task table, until eventually the cost factors are applied and final estimates obtained.

2.5.4 EDP-Estimator

Arrowood *et al.* have investigated knowledge-based EDP cost estimation with a prime motive to "provide less experienced project leaders with a tool to generate cost estimates and to explain reasoning processes."[29] Their goal is to implement a tool that is easy to use and that exploits the explanatory capabilities of expert systems. The EDP-Estimator pertains to 3GL development. It weighs four elements when making an estimate: labour costs, computer utilization, networking charges, and facility upgrades. These components were considered essential to determine the internal and external (contract) staffing needs. The EDP-Estimator at present is being implemented using Arity/PROLOG and Arity/Expert Development Package for use with IBM compatible microcomputers. The drawback with the EDP-Estimator is that it is tightly coupled with the knowledge-base, standards and procedures of a single organization. Also it does not have a basic model for estimation purposes and simply uses several heuristics such as:

> If procurement is minicomputer, cost is less than $50,000 and procurement exception is no, then add two worker-months of senior analyst time to labour of computing requirement (page 204).[29]

Such a strategy will probably limit its portability and use in other EDP shops.

2.5.5 Other Knowledge-Based Estimating Strategies

A recent exploratory study by Vicinanza, Mukhopadhyay, and Prietula[119] examines two basic issues: Is there expertise in software effort estimation? and can we use expertise to improve software effort? Their research strategy involved use of five experienced software project managers who served as expert subjects. Each manager was asked to sort a set of 37 commonly used project factors in order of importance, and to estimate historical projects given the size of such projects (eg., LOCs). Their results strongly suggest the existence of expertise in software effort estimation; with regards to techniques for improving effort estimation, they conclude:

> It has been suggested that a knowledge-based approach to the estimation problem may help improve the accuracy of existing models (see Ramsey and Basili)[120]. In support of this, our study indicates the particular form of reasoning that might be pursued is an analogical-reasoning approach.

Other interesting conlcusions:

- In most cases the managers had estimated effort better than two well established algorithmic models.
- Their research indicates that some cost factors do transcend organizational boundaries.

Ntuen and Mallik[30] have illustrated a general framework model for applying knowledge-based approach to cost estimating. It is however geared towards estimation in the engineering domain rather than the software industry. It provides a generic classification of the modelling tools for the cost engineer, and tabulates task descriptions for a model-based framework in a cost estimating expert system. Avots[31] has described the use of artificial intelligence techniques in the context of project management. The principal components of an expert system for schedule control were presented to illustrate analysis and predictive capabilities that could be added to existing project management tools.

2.5.6 Knowledge-based Systems for the Fourth Generation Problem

We can also use the knowledge-based systems approach to probe the impact of various project factors on the final estimates. Such a strategy will provide novice cost estimators and project managers with several benefits, such as an explanation capability, and the ability to simulate the impact of various project factors on the ultimate cost. A key objective here is to research a more open-ended approach to error correction. As new factors come into play, they can be integrated with the expert system to provide better estimates. This aspect of the 4GT Model is discussed in detail in Chapter 4. Knowledge-based techniques can also play a useful role in the initial planning process that precedes cost estimation. Some empirical evidence[32] for this was provided in the context of assisting the manager with selecting a methodology for software development and deciding if prototyping was an appropriate choice.

Chapter 3

The 4GT Estimation Model

3.1 Introduction

In this chapter we introduce the 4GT estimation model. The scope of the model is 4GT effort estimation, that is when software applications are developed using fourth generation tools such as report writers, form generators, and application generators. Basically we are interested in the implementation of small to medium business information systems by programmers or end-users. The model at present does not consider larger applications within its scope as such applications usually involve extensive procedural and/or non-procedural coding. The following categories of products fall within the scope: Personal Computer Tools such as dBASE IV, Query Languages and Report Generators eg, QBE, RPG, Graphics Generators eg, SAS Graph, Application Generators, eg, Oracle.[2]

The proposed model can be used to estimate application development effort with different 4GT paradigms. (This Chapter introduces different 4GT development paradigms and Chapter 5 describes their effort estimation.) Only theoretical details of the model are presented in this chapter. Model experimentation, calibration, usage, and validation are described in Chapter 5.

3.1.1 Model History

The *4GT Model* was designed and developed on the basis of interviews with practitioners, literature research, product study, and project data collected between 1988 and 1991. Personal project management experience as well as discussions with members of the thesis committee also played a major role in the model development.

At the very outset it was recognized that a general purpose model for cost estimating all types of fourth-generation development was difficult to develop (largely due to the wide diversity of fourth generation languages). It was subsequently decided that only fourth generation tools (4GTs) should be modeled. The objective was to create a meta-model[69] for estimating effort involved with 4GT development. Using the meta-model as a basis, organizations could either adapt or create their own cost model in accordance with their own environments. DeMarco and Lister concur with such a strategy; they state that, "cost models do work, but they have to be made local to the environment in order to provide useful forecasts of development time and effort".[85]

The model evolved through two major stages:

- an *initial* prototype stage in which the *effort estimation* and *effort adjustment* components were designed and developed, and

- an *installation* stage in which the effort estimation component was enhanced, re-calibrated and installed in a commercial setting for experimentation purposes.

Details pertaining to the above stages are presented below.

3.1.2 The Initial Stage

The initial version of the model was based largely on literature review, Fourth Generation product review, and personal experience obtained as project manager of several projects. Experimental data gathered during the implementation of the following projects between 1988-90 (under my supervision) proved to be useful for gathering data and conducting research related to the initial version of the model.

1) *Spatial Accounting Database* project for Physical Plant, University of Winnipeg.[88]
2) *Trackers Record System* project for the V.P. Admin Office, University of Winnipeg.[89]
3) *Client and Applicant Tracking System* project for Mayday Personnel, Inc., Winnipeg.[90]
4) *Weights and Measures Microcomputer System* project for Consumer and Corporate Affairs (Canada), Winnipeg.[91]

The basic design of the model was exposed to criticism at various seminars and revised as needed[32,70-78,92,93]; it was also demonstrated to practitioners in Winnipeg.[77,84] This exposure at various sites strengthened the model considerably.

3.1.3 Experimenting with the Model

With real-world experimentation of the 4GT Model in mind, Ted Janzen, Associate Manager Computer Systems, Great-West Life, was approached in May 1991 (on recommendation from Kerry Morris, Systems Analyst, Computer Systems Group, Great-West Life). Janzen's Computer Systems group was actively developing 4GT based application systems using ORACLE and they were therefore quite interested in installing the 4GT Model for effort estimation purposes. Great-West Life was suitable as a site for experimentation with the Model as:

- the state of art development was taking place using fourth-generation tools, and
- relevant metrics data from on-going or past projects were readily available.

Largely on the basis of intensive interaction with Ted Janzen, and Wendy Smith (Senior Systems Analyst at Great-West Life), the effort estimation component of the 4GT Model was enhanced and made easier to use.

A formal approach[68] was used to interview and document data. Before our very first meeting, a report describing the existing 4GT Model was given to the interviewees at Great-West Life (see Table 3.1). The initial meetings served the purpose of acquainting participants with the 4GT Model and also informing us about their project management and cost estimation practices. Most of the participants had estimated ORACLE projects before and were quite familiar with the topic of cost estimation. Actually, their current practice involved submission of project effort estimates by all team members (to the project manager).

More than twenty five meetings took place between May 1 and October 1991 (see references 77 through 84). Each lasted more than one hour but usually less than three. Both group meetings and one-on-one meetings took place. A bulk of the information, however, was gathered during one-on-one meetings. It was also necessary to communicate and gather information using the telephone and fax machine; many such communications took place.

The initial few meetings highlighted the difficulty of calibrating the 4GT Model at the Great-West Life. For instance, the practitioners indicated that it was difficult to calibrate and document all relevant specification operations (SEs). They also felt that estimating effort using the model could also be tedious as there were too many SEs (one for each unique specification operation).

After further discussion we decided to make the following two changes to the model:

- placing *similar SEs* into one category

  — this revision simplified the model considerably as we now had to calibrate weights for each *SE category* only (and not for each SE as was required before).

- tying each SE to a *screen field* instead of a *screen page*

  — this resolved some problems with the model. For instance, now it did not matter if a form had more than sixteen screen fields per screen page or less than six per page. Obviously more specification effort is required to develop a form with twenty screen fields as opposed to just two screen fields. (Note that a magnitude correction table was provided with the initial version of the model but it did not extrapolate beyond sixteen screen fields per page.)

Both the above strategies simplified the model considerably. For instance, it was now easy to count SEs — all one had to do now was to locate screen fields!

Having made the above enhancements to the model the following key activities took place between June 1991 and November 1991: (a) data collection

Table 3.1: Resources Used to Develop the 4GT Model

Name	Title	Organization
Norbert Kaehler	Assistant Manager	Investors Group Development Services, Information Systems and D.P.
Annegret Layer	Systems Analyst	Investors Group
Irene Warkentin	Sr. Computer Systems Specialist	Great-West Life, Computer Systems Group.
Calvin Trainor	Sr. Computer Systems Analyst	Great-West Life, Computer Systems Group.
Ted Janzen	Associate Manager	Great-West Life, Computer Systems Group.
Wendy Smith	Senior Systems Analyst	Great-West Life Computer Systems Group.
Rob Buskens	Systems Analyst	Great-West Life, Computer Systems Group.
Allison Minaker	Project Manager	Great-West Life, Computer Systems Group.
Mavis Hildebrand	Systems Co-ordinator	Pitblados & Hoskins (and earlier on at Great-West Life).

to calibrate the weights for each SE category; (b) determination of the expansion factor; (c) validation of the entire 4GT model. All data collection and classification into SE categories were performed independently by Smith. The results are presented in Chapter 5 and also published elsewhere.[86,87]

Quite independently, comparable experimentation with the 4GT Model also took place at the University of the Winnipeg using ORACLE as well.[82] Results obtained here provided us an opportunity to validate the Great-West Life model weights and to investigate portability issues (see Chapter 5 for details).

3.2 Measuring Application Effort

In this section we describe the systems development life cycle and the different development paradigms associated with it when 4GTs are used. As we are interested in developing a model that measures application system size we address the following issues:

(a) What is the nature of the life cycle when 4GTs are used?

(b) What functions adequately represent system size and effort when 4GT development occurs?

(c) Is there a good predictor for modelling the above functions?

Section 3.3 deals with the issue of systems development life cycle; section 3.4 identifies functions representing 4GT development effort; section 3.5 deals with the issue of predictors; and section 3.6 presents the 4GT model for measuring application system effort.

3.3 Systems Development Methodology

Table 3.1 illustrates the systems development life cycle for developing 4GT business information systems. This methodology is similar to life cycles described

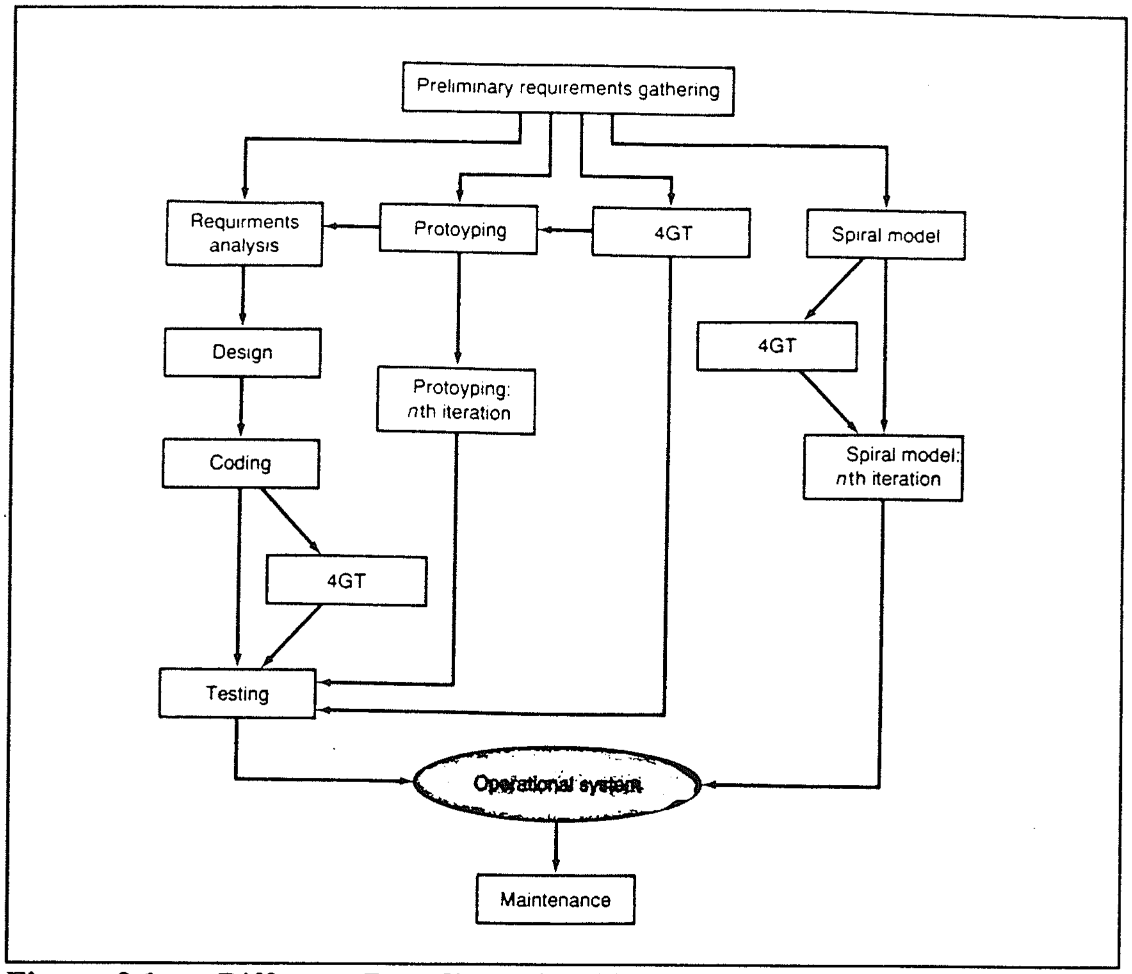


Figure 3.1: Different Paradigms for 4GT Development

in various software engineering and systems analysis texts [6,62,63,64]. The introduction of 4GTs, however, has made it possible for us to adapt this life cycle. The most notable of which is a prototyping-based systems development life cycle. Figure 3.1 reproduced from Pressman's Software Engineering text book[6] illustrates different software development paradigms now possible due to the introduction of 4GTs. We will discuss these paradigms under two categories: "the traditional approach", and "the prototyping approach".

3.3.1 The Traditional Approach

This essentially is the traditional systems development life cycle. Table 3.2 illustrates the steps involved with such a life cycle. Details of what occurs at each stage is presented briefly below:

Table 3.2: Steps Involved in Traditional Life Cycle

Phase I	Feasibility Study and Requirements Definition
Phase II	General Analysis, Design, and Data Modelling
Phase III	Detailed Design
Phase IV	Coding
Phase V	Testing,

Phase I

Feasibility Study and Requirements Definition

The feasibility and scope of the software development project are investigated. Very general information about the users needs is available at this stage.

Phase II

General Analysis, Design and Data Modelling:

A general system study, analysis, and design take place at this stage. It entails documenting the complete system — generally by creating data flow diagrams. The analysis stage is summarized by Fertuck as follows (page 6):[62]

> The analysis stage is actually a learning process in which the analyst tries to gain an understanding of what the user does. The Data Flow Diagram is an intermediate product that allows the analyst and the user to communicate unambiguously. It summarizes the information that the analyst needs during the design stage. It does it in a clear graphic way that the user can understand.... the final result of this stage is an understanding of the system documented by Data Flow Diagrams.

Data modelling also occurs at this stage. As part of the process of data modelling we identify user views and reports, normalize them into tables, and create a conceptual data model. Input/output layouts (forms and reports) of the system are designed and all major process modules are identified at this stage.

Using CASE or equivalent tools, the following products are created at this stage:

- Entity Relationship Model, Data Flow Diagram, and Program Structure Charts.
- A Data Model (with tables in 3NF).
- Screen Layouts, Report Formats, and Processes.

Phase III

Detailed Design:

The objective of the detailed design phase is to refine the tasks of the previous phase, and to translate requirements into a "representation of the software that can be assessed for quality before coding begins".[6] Here we perform additional data analysis, re-normalize the tables, and improve the logical and conceptual models, if necessary. The physical aspects of the database can now be designed — this includes designing stored record formats, selecting access method, and determining the blocking factor. A major deliverable at this stage is documentation of application design (covering all modules).

Phase IV

Coding:

At this stage we create the software system. Coding and code-generation takes place using 4GTs (procedural and non-procedural languages are used if necessary). A users manual and an operations manual are developed.

Phase V

Testing:

Coding and code-generation takes place one more time; module testing, system testing, and integration testing take place and the system is demonstrated to the user one more time for acceptance. If everything is satisfactory, documentation is generated, and conversion takes place.

3.3.2 Prototyping Approach

Prototyping can be defined as a process that enables the developer to create a working model of the software that must be built.[6] Some organizations intentionally implement only a single version of the prototype and discard it subsequently on obtaining an initial understanding of the users needs. The terms *throwaway, explorative, experimental,* or *non-evolutionary prototyping* are all often used to characterize this practice. In contrast, *rapid prototyping* or *evolutionary prototyping* allows several increasingly refined versions of the prototypes to exist. (Actually, the same prototype is revised and enhanced at the end of each phase of the systems development life cycle.)

3.3.2.1 Non-Evolutionary Prototypes

Throw away prototyping is practised by individuals who believe that prototypes must be discarded as they are of poor quality. They cite reasons such as: a) prototypes are developed in a hurry; and b) prototyping short-circuits the various checks and balances a systems development life cycle has to offer. Table 3.3 describes the various stages in the systems development life cycle when a non-evolutionary prototyping paradigm is used.

Table 3.3: Steps Involved in Throwaway Prototyping

Phase I	Feasibility Study and Requirements Definition
Phase II	General Analysis, Design, and Data Modelling: Prototyping
Phase III	Detailed Design
Phase IV	Coding
Phase V	Testing

Phase I, III, IV, and V are similar to the traditional life cycle. Therefore only the extensions are described below:

Phase II

General Analysis, Design, and Data Modelling: Prototype Development

The information and data gathered during general analysis, design, and data modelling are usually sufficient enough to begin prototyping. We first develop an *initial version* of the prototype on the basis of available information from the users. This version is demonstrated to all key users, and prototype enhancements are sought. A few more iterations of the prototype occur subsequently, and we come up with the *final version* of the prototype when the users are happy with what they see. The following tools are used for prototyping: RDBMS, 4GL, screen generator, form generator, report generator, menu generator, and application generator.

3.3.2.2 Evolutionary Prototyping

The availability of 4GLs and 4GTs has made evolutionary prototyping very appealing today as good quality code (i.e., prototype) can be generated even in a hurry. Several practitioners therefore see no merit in discarding a prototype under such circumstances. They claim that since the bulk of the code was automatically generated, it is of good quality (in the sense that human introduced errors do not exist, and not in the sense that the code is efficient — which is debatable).

The evolutionary prototyping methodology is similar to the non-evolutionary methodology (see Table 3.3 for details). The significant difference is that a prototype created during the second phase evolves even further as the systems development life cycle progresses. At the end of each phase, newer revised versions of the prototype are produced. The final phase produces an operational prototype, which is delivered to the user.

The key advantage with the above methodology is that it favours user involvement at all stages of the life cycle and not just at the start; development risk is therefore reduced considerably. Coding effort might be reduced noticeably during the implementation phase as the prototype needs only to be extended and not re-built from scratch — as in the case of the throwaway prototyping.

Table 3.4: Steps Involved in Evolutionary Prototyping

Phase I	Feasibility Study and Requirements Definition
Phase II	General Analysis, Design, and Data Modelling: Initial Prototype Development
Phase III	Detailed Design: Prototype Enhancement
Phase IV	Coding: Prototype Enhancement
Phase V	Testing: Operational Prototype

Figure 3.1 also indicates that there is a *spiral model* alternative to software development. This model focuses on risk analysis, and follows the evolutionary prototyping methodology very closely.

3.3.3 Conclusion

The preceding discussion introduces the systems development life cycle in the context of measuring 4GT development effort. Figure 3.1 provides us a basis for understanding the differences between the different systems development life cycles. In order to understand the different options illustrated in the figure it is mapped using numbers and re-displayed in Figure 3.2.

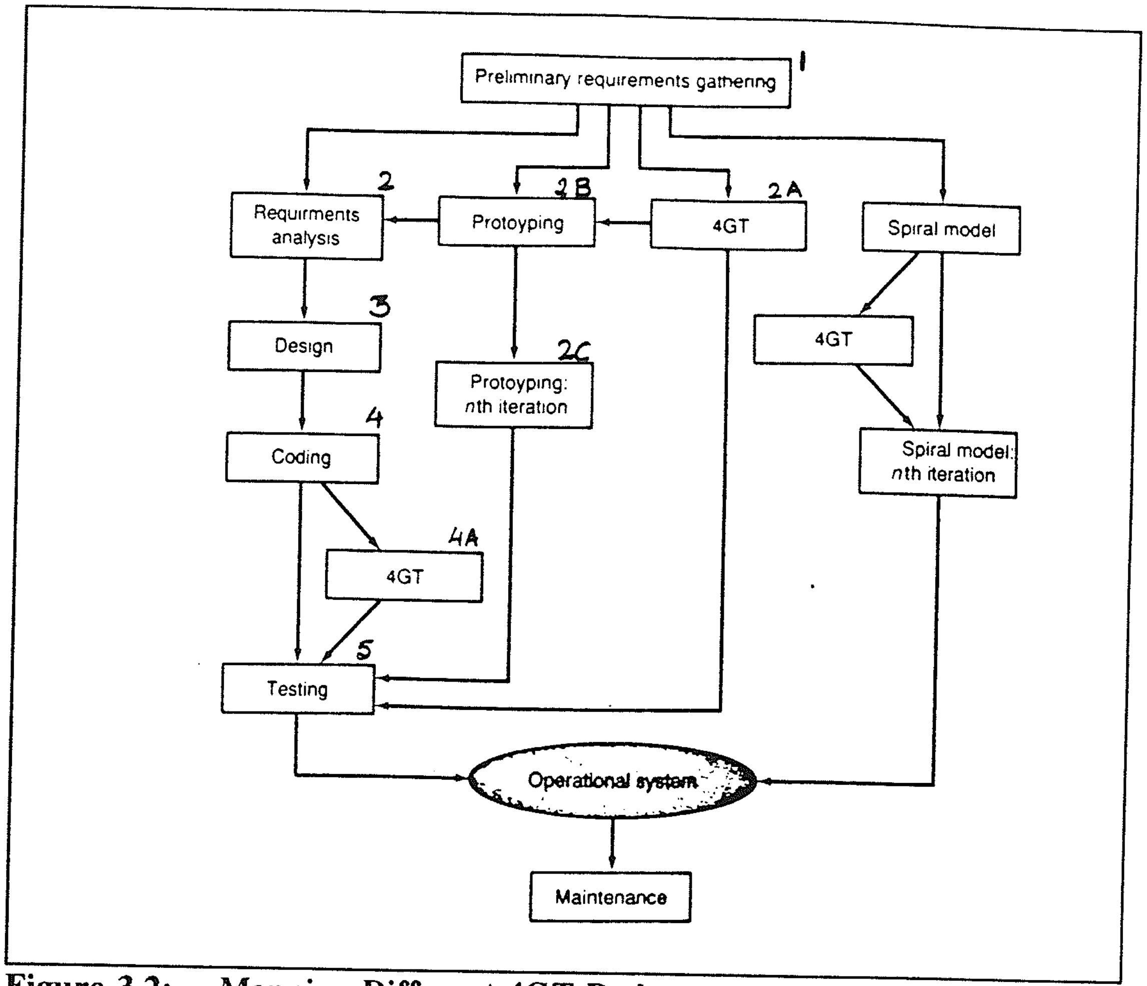


Figure 3.2: Mapping Different 4GT Paths

We notice that it portrays the following different paths (and hence different paradigms):

- 1, 2, 3, 4, 5 represents the traditional life cycle when no 4GTs are used.
- 1, 2, 3, 4, 4a, 5 represents the traditional life cycle with usage of 4GTs.
- 1, 2a, 2b, 2, 3, 4, 4a, 5 represents the non-evolutionary (throw-away) prototyping approach.
- 1, 2b, 2c, 5 represents the evolutionary (rapid) prototyping approach.

Note that we ignore the paths associated with the spiral model life cycle as they are similar to the evolutionary prototyping approach. The path: 1, 2a, 5 is also ignored as it is too futuristic. Commenting on this aspect, Pressman states:[6]

> Ideally, the customer would describe requirements and these would directly be translated into an operational prototype. But this is unworkable. The customer may be unsure of what is required, may be ambiguous in specifying facts that are known, and may be unwilling to specify information in a manner that a 4GT tool can consume. In addition, current 4GT tools are not sophisticated enough truly "natural language" and won't be for some time.

Ignoring the path: 1, 2, 3, 4, 5 where no 4GTs are involved we now recognize the following distinct paradigms for 4GT based development: a) traditional 4GT life cycle without prototyping; b) evolutionary prototyping using 4GTs; and c) non-evolutionary prototyping using 4GTs.

The 4GT Model as described in this thesis pertains primarily to the evolutionary prototyping paradigm, i.e., most of the research and data collected relate to this paradigm. Focusing on this paradigm is worthwhile as it is the most dominant 4GT life cycle today. Actually "fourth generation technology" and "prototyping" go together — Clarke includes "prototyping capability" in his list of five "defining characteristics of the fourth generation environment" (p. 25)[66]. With 4GT based application development this is a very natural paradigm as prototyping is easily facilitated. Regardless, the model introduced in this thesis is flexible and supports various paradigms. This aspect is presented in Chapter 5.

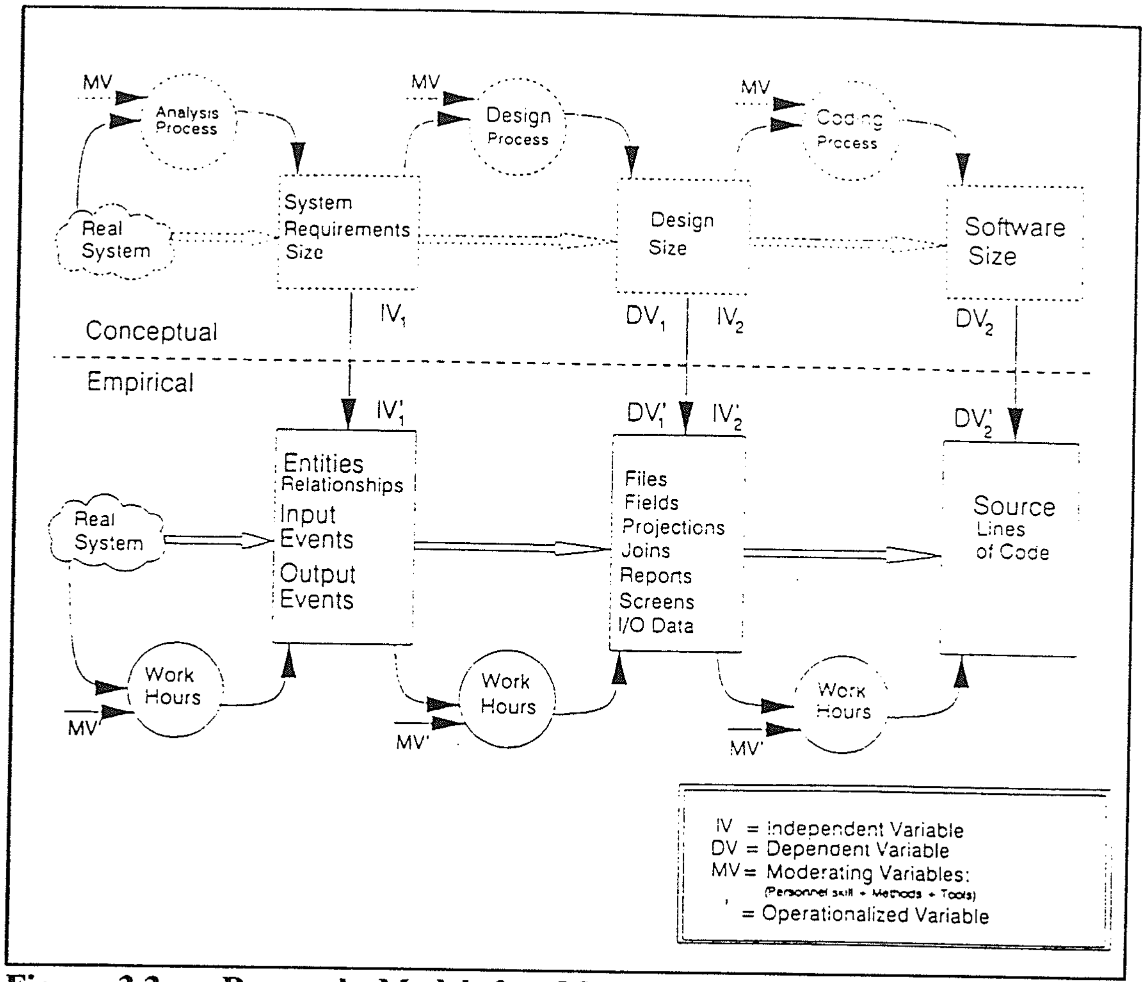


Figure 3.3: Research Model for Measuring Information System Size (Wrigley & Dexter)

3.4 Representing System Size - Theoretical Issues

In this section we deal with the theoretical issue of representing system size using functions. Our modelling approach is based on software *functional decomposition* techniques that occur during systems development. Functional decomposition is a stepwise elaboration mechanism for refining the processing tasks that are required for software to accomplish some desired function.[6] There are different approaches to decomposition (a recent approach being object-oriented analysis) — the overall strategy here is to define and characterize the software design in terms of a software functional framework.

There is a relationship between such a structural framework and the eventual cost of a system, and most effort estimation methods or models are based on this hypothesis. Theoretical research conducted by Silver agrees with the above fundamental strategy:[103,104]

> The essential conclusion reached is that software functional decompositions (to an appropriate level of depth) may indeed serve as the basic underpinnings for the design activity and associated cost/performance specification even at the requirements level, and indeed throughout the entire software development life cycle.

A formal research model illustrating the theoretical relationships among the measures of system size available at each phase of system development — analysis, design and coding was introduced recently by Wrigley and Dexter;[26] this model, illustrated in Figure 3.3, is described by them as follows:

> At the conceptual level, there exists a relationship between the size of the requirements of the real system and the size of the eventual software product. Each stage of development is achieved through the various processes; analysis, design, and coding. System specifications at each stage are transformed into the next stage through these processes...

They conclude that as "user requirements are temporally antecedent to the eventual delivered software, one can specify that a causal relationship exists between actual requirements, design and coding effort".[26] We next describe our model as it pertains to mapping of system requirements with design.

3.4.1 Input, Output, and Process Tasks

Useful information about application software functionality, and hence required functions, is usually available from the user requirements document, produced at the end of the requirements definition stage. This document might be incomplete and not specific enough to identify all functions. But by the next stage, (general analysis, design and data modelling) the proposed system's outputs, inputs, and processes are fully identified.

Literature survey reveals that several function-oriented cost modelling researchers have consistently considered use of system input, output, files, and processes for measuring and modelling a business information processing system's size and effort. For example, Symons MARK II model assumes that transaction-oriented systems consist of logical input/output/process combinations.[38] Albrecht's FPA approach, introduced in Chapter 2, uses outputs, inputs, files, inquiries, and interfaces. (Note that interfaces can be treated as a project factor, and inquiries can be viewed as a combination of input/output/process.)[38]

Parkin[65] describes a US Army function-oriented model that estimates total project person-months using outputs, number of record types in database (which is comparable to using the number of tables), number of files, and number of input transactions.

Itakura and Takayanagi's[67] model for estimating development effort of COBOL projects includes the following functions — input/output files, reports and processes.

3.4.2 Form, Report, Data, and Process Functions

As evident from some of the above models, inputs, outputs, and processes can be viewed in terms of aggregates (such as form, report and process functions). For instance, a form function type can be viewed as a combination of an input, process, query, and output. This is especially natural for application development in the fourth-generation environment. For 4GT applications it is probably convenient to view the system in terms of forms, reports and process functions rather than input, output and process tasks. Users actually are more comfortable identifying forms and reports they want than specifying input, output and process tasks for an application.

The following function types are being introduced for the purposes of effort estimation:

Form

In order to interface with the database we design and implement forms. Forms are also part of the delivered application system. Forms consist of screens and are used for the following purposes:

Input	—	use the form to enter new data
Query	—	use the form to interrogate stored data
Output	—	uses the form to display queried data
Update	—	use the form to change data
Process	—	use the form to implement transaction logic

A forms-generator or screen-generator can be regarded as a source for the form function. Albrecht's original paper on FPA describes "data form" as an example of input function, "printed report" as an example of output function, and "disk files, tape files, and input card files" as an example of file function. (Of course, today data forms can no longer be classified as an input function only, and also report functions today allow substantial customization via queries before they are printed.)

Report

In order to interface with the database we design and implement reports. Forms are also part of the delivered application system. Reports are similar to forms but their primary purpose is to display the results of a query or simply display data stored in the application database. Subsequently they allow the following actions:

Output —print data on the screen, or through a printer

Query — limited interrogation of the stored database is allowed with the intention of creating customized output.

A report-generator can be regarded as a source for the form function.

Data

This function type refers to database tables and files of the proposed application. Just like forms and reports it is a deliverable — part and parcel of the software system given to the users. Most transaction processing systems today and especially 4GT-based systems use the database approach for systems development. Such an approach concentrates on understanding, using, and documenting data. According to Fertuck, "by viewing the database as the central component, the analyst concentrates on defining the data correctly. Later the analyst worries about inputs, reports, or programs that transform the data in the database."[62]

The database tables associated with the project are usually identified as early on as the *preliminary design* stage via the *entity relationship diagram (ERD)*, and later on during the *general analysis, design and data modelling* stage via the *data model*. Data Flow Diagrams and its *data stores* are also associated with the data function. DeMarco defines data stores as follows (page 5):[61]

> Data stores can be thought of as manual or automated files or databases or any other accumulation of data.

ERD's, data models and DFD's can therefore be regarded as sources for the data function.

Process

Process function type represents procedures and modules that will eventually be coded in a procedural or non-procedural language. (4GT form- or report generators are not used.) Process functions are similar to forms and reports and they can be associated with the following tasks:

Input	— enter new data
Query	— interrogate stored data
Update	— change data
Process	— implement programming logic

Procedures or modules performing complex calculations and algorithms are generally classified as a process function type. Note that a form or report also could be classified as a process function if a form- or report generator cannot be used (due to lack of tool functionality). The bulk of our current modelling effort is only on form, report, and data function types (conventional cost models do not address them adequately).

3.4.3 Conclusion

In this section we described how information system can be measured in terms of the following function types: form, report, data, and process. We also showed that this hypothesis has been well researched. For instance, Wrigley and Dexter researching a general strategy for measurement and evaluation of systems development environments conclude that screens, reports, and files (derived during the preliminary design stage) explain information system size very well. Their research and analysis involved FOCUS projects which is a 4GL. They also indicate that with data-strong systems (i.e., database oriented application systems) measurement in terms of reports, screens, etc., is sufficient. (However, for scientific applications it is likely that measurement units in addition to screens, reports, etc., would be needed to reflect design and requirements size satisfactorily.[59])

Finally, we note that various key products of the systems development life cycle can be identified in terms of the function types introduced in this section.

This is summarized below:

Product	**Function Types**
Entity Relation Diagram	Entities represent tables & files (data type).
Data Model	Consists of tables, and files (data).
Interfaces	Inputs and outputs via forms, reports, and process types.
Data Flow Diagrams	Data stores (data type), process type.
Prototype	Forms, reports and process types.

3.5 Predictors

In this section we describe prevailing predictors and introduce our new predictor for the 4GT-based software development environment. According to DeMarco (page 54),[1]

> Every metric falls into one of two categories: either a "result" or a "predictor". A *result* is a metric of observed cost, scope, or complexity of a completed system. Examples include total cost, total manpower, elapsed time, or cost or manpower.... A *predictor* is an early-noted metric that has a strong correlation to some later results.

Size measures such as LOC, Function Point, and code volume can be regarded as examples of predictors.

3.5.1 Prevailing Predictors

We recount the limitations of the two prevailing predictors of the 3GL environment — LOC, and Function Point (FP) in this section. Let us consider the

LOC metric first. We note the following problems when it is used as a predictor for estimating effort in the 4GT environment:

- There is no standard definition for a line of 4GL code.[33] It is therefore difficult to estimate the number of lines of 4GL code, and difficult to convert a 4GL LOC to a 3GL LOC (or vice versa). Such conversions are required by existing cost models.
- In the case of 4GTs, especially, it is not practical to use LOC as most of the code is generated automatically, without any substantial programming. Fairly complicated forms and reports are generated by simply specifying the functions. It is therefore difficult to collect true project statistics in terms of "effort put in" and "LOCs developed".
- While some generators display the source code generated for documentation or editing purposes, many do not display any code at all. For example dBASE III's report generator produces executable code directly and we cannot see the code generated (hence we are unable to count the LOC).
- 4GLs produce substantially smaller code sizes when compared with 3GLs.[34] Consider the fact that while data declarations are almost non-existent in 4GLs they typically constitute more than half the code for a COBOL program.[35] Such simplicity in computing should be reflected in effort estimates produced by a model but cannot be done so easily.

In conclusion, we note that historically, cost models using LOC as a predictor (such as COCOMO) were designed around databases consisting of 3GL projects only. Fourth generation tools, as we know them today, did not exist then. Obviously, such models or their predictors are not very suitable for estimating 4GT projects.

As an alternative to using LOC, let us consider the use of predictor FP (as defined in Albrecht's FPA Model). FP eliminates the need to estimate in LOCs thereby eliminating several paradoxes caused by LOC measures.[36] It is therefore more suitable than LOC for estimating fourth-generation development effort. The FPA method, however, has limitations:

- The resolution capability of each FP is inadequate when applied to application development using 4GTs. Consider the following comments from Dreger with regards to counting FPs when a report or screen generator is used:[37]

  > ...if this tool (form or report generator) is provided it is impossible to count Function Points for all possibilities, and you must settle the issue by crediting the application with one complex output (FP).

  Suggesting a default count of one complex FP for all forms and reports generated is certainly inadequate as some forms are significantly more effort-consuming than the others.

- The weighting associated with the FPA method is also inadequate for 4GT-based development (where a lot of specification involving data elements occur). For example, Symons questions why "a system component containing, say, over 100 data elements is given at most twice the points of a component with only one data element". This can result in a significant effort estimation error for 4GT-based development. (The 4GT Model resolves this problem elegantly).

Several other problems specific to the FPA model are documented by Symons in his paper conveying the impression that the model is far from perfection for either 3GL or 4GL based development (pp. 1-8).[38] Especially for 4GT development we regard the model as deficient — as with LOC-oriented models, this can be attributed to the fact that the FPA method was designed around databases consisting of largely 3GL projects only — powerful fourth-generation generators as we know them today, did not exist then.

3.5.2 Attributes of a New 4GT Predictor

In section 3.4 we examined the nature of software development and concluded that at a global level information system size could be represented using form, report, data, and process functions. However, we need a predictor to estimate the effort involved with developing each function, as well. As we established that both LOC and FP are inadequate predictors for 4GT-based development (in the preceding section), we now describe a suitable predictor for our purposes.

A very important characteristic of application development using form-generators, report-generators, and application-generators is that:

a) we generally *specify* what is to be accomplished, and

b) most of the specification effort is focused on and around *screen fields*.

The term *screen field* refers to the data elements of form, report, data, and process functions. The screen field plays a very important role in the specification-oriented programming process used by 4GTs as the entire foundation of default logic is built in and around and them. This belief is supported by ORACLE (page 1.3)[55] and is evident in several other 4GTs as well (such as ORACLE's SQL*FORM;[57] UNISYS's ACCEL;[100] and Relational Technology's Ingres[101]). All such 4GTs, including low-level 4GTs such as dBASE IV,[102] require us first to define skeletal screen fields for each form and report, and then to implement logic around them — usually with specifications or macros or code.

In view of the above, we can regard the following as being essential attributes of any 4GT predictor:

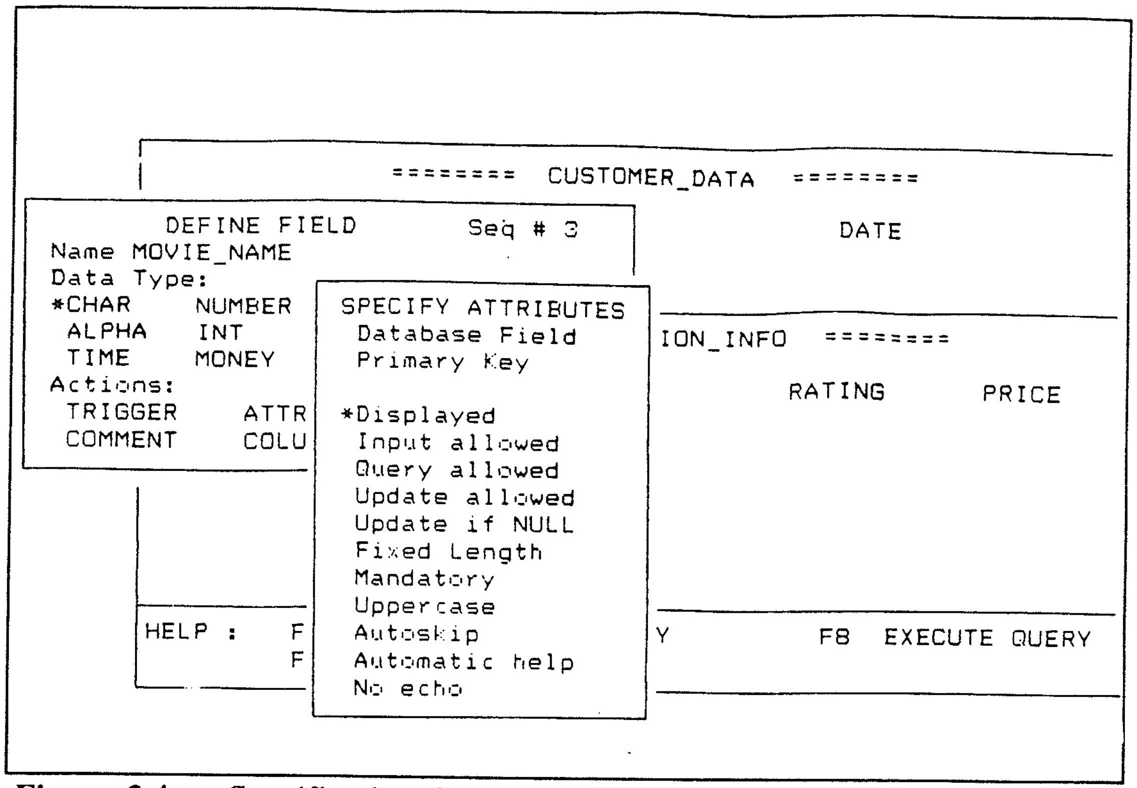
- support of the specification oriented programming paradigm of 4GTs (as used by form-, report-, and application generators), and
- use of the screen field as an effort parameter.

The *Specification Element* (SE) is being proposed as a predictor that addresses the above issues. The term "specification element" is a hybrid of the two terms *specification*, and *screen- or data element*. An SE can therefore be defined as "a specification task associated with implementing a screen field or a data element". Examples of an SE are "automatically retrieving name and address on entering ID" and "converting any input data automatically to upper case". Figure 3.4 below illustrates specification of this SE; note that the SPECIFY ATTRIBUTES window with the *uppercase* option is pulled open.

Each SE has an effort value (in person-hours) that represents the effort required to implement it. Collectively SEs represent the functionality and effort required to develop an entire software application.

3.5.3 Categorizing SEs

To facilitate practical use of SEs as a predictor we must first classify them unambiguously. A study of 4GTs, design and development of applications using them, and interactions with practitioners provide us with a basis for categorizing SEs[2,3,55,57,78,79,100,101,102,105]. ORACLE is used as a case study here to illustrate the



```
                    ========  CUSTOMER_DATA  ========
 DEFINE FIELD          Seq # 3                       DATE
Name MOVIE_NAME
Data Type:
*CHAR     NUMBER  SPECIFY ATTRIBUTES
 ALPHA    INT      Database Field     ION_INFO  ========
 TIME     MONEY    Primary Key
Actions:                                     RATING      PRICE
 TRIGGER     ATTR *Displayed
 COMMENT     COLU  Input allowed
                   Query allowed
                   Update allowed
                   Update if NULL
                   Fixed Length
                   Mandatory
                   Uppercase
HELP :      F      Autoskip           Y        F8  EXECUTE QUERY
            F      Automatic help
                   No echo
```

Figure 3.4: **Specification Operation using Oracle**

illustrate the distinct levels of form design and development that occurs with 4GTs.

3.5.3.1 Form SEs

As indicated earlier on, specification operations involving screen fields play an important role in application development with 4GTs. Distinct development stages and discrete specification categories are also evident (page 6-12),[55] (page 8-2).[57] This fact is used to identify the following distinct categories of SEs:

- *Simple SEs*

  They account for the nominal effort involved in implementing a skeletal form.

- *Basic SEs*

  They are associated with use of simple specification operations.

- *Detailed SEs*

  They are associated with sophisticated specification actions — such as those involving use of macros, or triggers.

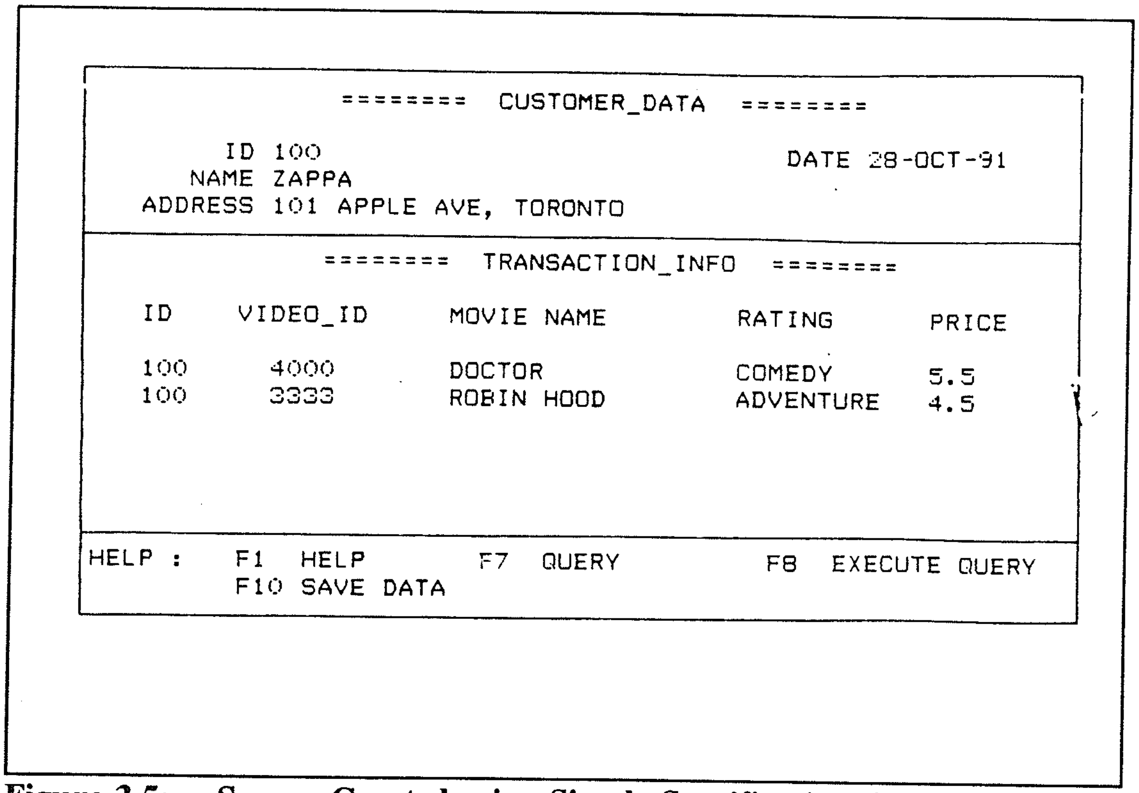
- *User exit*

  User exits are associated with procedures or programs written in a conventional programming languages

Simple SEs

Screens consisting of screen fields have to be created first. We can provide logic to a form only after the initial screen structure is implemented. This activity is described by ORACLE as the first level of form design where one creates a "screen consisting of fields without any special validation or enhancement" (page 1-8).[57] As very elementary specification operations are involved with screen design we classify the specification elements of this category as "Simple SEs". SEs in this category are collectively responsible for creating (and altering) an initial form. Typical activities involved here include: relabeling fields, modification of field sequence numbers, cutting and pasting of fields, and visual enhancement of form.

All applications require a certain amount of such specification activities. The end result is a form with skeletal screen fields such as the one illustrated in Figure 3.5. (It must be noted here that screen prototyping — the most simple type of prototyping — involves development of several such screens. Hicks describes this activity as follows (p.196):[118] "The system analyst and the user quickly generate a **skeleton application program** which serves as a model for the application. The end-user can interact .. and thereby refine the system's requirements".)



```
                ========  CUSTOMER_DATA  ========

     ID 100                                       DATE 28-OCT-91
   NAME ZAPPA
ADDRESS 101 APPLE AVE, TORONTO

               ========  TRANSACTION_INFO  ========

ID      VIDEO_ID      MOVIE NAME          RATING        PRICE

100       4000        DOCTOR              COMEDY        5.5
100       3333        ROBIN HOOD          ADVENTURE     4.5

HELP :    F1  HELP        F7  QUERY           F8  EXECUTE QUERY
          F10 SAVE DATA
```

Figure 3.5: Screen Created using Simple Specification Operations.

Basic SEs

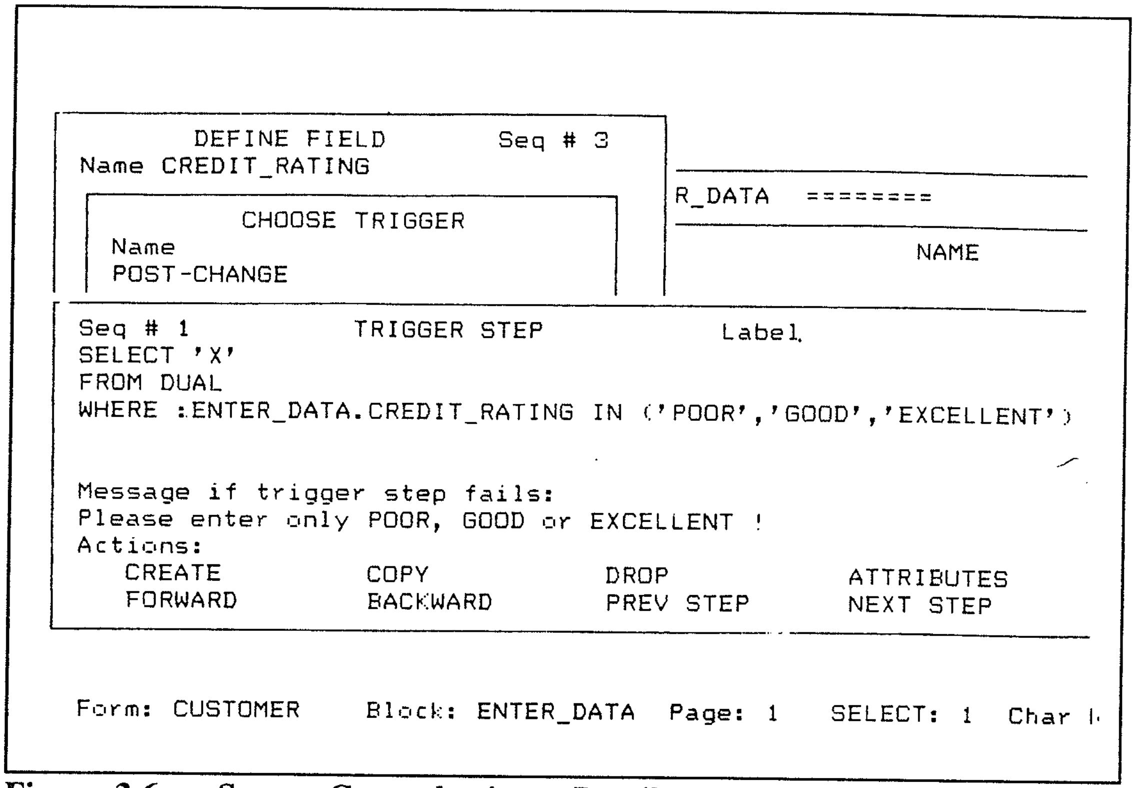
Some of the above screen fields will require some basic validation and assistance. This can be regarded as the second of the them ashree levels of forms design. At this stage one can "specify field ranges, default values, and help messages by making a single entry on a SQL*Forms window" (page 1-8).[57] Since SEs in this category involve single action steps or specifications, they are collectively referred to as Basic SEs. Examples of such SEs include: specifying upper and lower field ranges, and automatically converting data to uppercase.

Detailed SEs

Some of the screen fields will require more "sophisticated validation and assistance by writing triggers or short sequences of SQL or SQL*Forms commands" (page 1-8).[57] Triggers are macros or commands that are activated when certain fields are used. We classify triggers and related sophisticated specification operations as "Detailed SEs". The following are examples of Detailed SEs:

- validate an entry against a list of values in a column of a table.
- retrieve a product name and list price from a table when the operator enters the product code number.
- calculate the total amount of an item ordered from its quantity and price fields.
- assure that an actual price is discounted no more than 20% off list.

The detailed specification operation, "restrict fields to a set of data values" is illustrated in Figure 3.6.



```
          DEFINE FIELD          Seq # 3
   Name CREDIT_RATING
                                          R_DATA    ========
               CHOOSE TRIGGER
       Name                                                       NAME
       POST-CHANGE

   Seq # 1            TRIGGER STEP              Label
   SELECT 'X'
   FROM DUAL
   WHERE :ENTER_DATA.CREDIT_RATING IN ('POOR','GOOD','EXCELLENT')

   Message if trigger step fails:
   Please enter only POOR, GOOD or EXCELLENT !
   Actions:
      CREATE           COPY             DROP             ATTRIBUTES
      FORWARD          BACKWARD         PREV STEP        NEXT STEP

   Form: CUSTOMER     Block: ENTER_DATA   Page: 1    SELECT: 1   Char I
```

Figure 3.6: Screen Created using a Detailed Specification Operation.

User Exits

Finally, we describe a distinct category by itself — the *user exits*. Some screen field triggers can be implemented in such a way as to permit a temporary exit to 3GL routines. Languages such as C are commonly used to implement user exits. User exits are sometimes necessary if faster response time is required or if the desired functionality cannot be provided by a 4GT. Hicks explains this distinct feature of 4GTs as follows: "Each application is likely to have unique requirements. Therefore most application generators contain *user exits*. User exits allow a user or a programmer to insert program code that takes care of these unique requirements of the application". ORACLE explains it more technically as:

> A trigger step can temporarily exit SQL*Forms to a program written by you or other users. You can use such user exits to process information in tables and form fields, display messages, and perform many kinds of processing that are beyond the scope of SQL and SQL*Forms (page 9-28).[57]

Obviously, user exits require significantly higher effort than the other categories described. User exits are not necessary with simple application systems. End-users will typically not be asked to implement them as they require traditional programming skills.

User exits to 3GL procedures from within a 4GT are not possible always. For example, ORACLE's SQL*Report Writer does not permit user exits. This category then is not relevant here. Such SEs are not identified when effort estimation occurs with such tools.

3.5.3.2 Report Form SEs

Report form screen fields are similar to form screen fields and therefore have the same distinct categories — simple, basic, detailed, and user exit — as described in the preceding section.

3.5.3.3 Data Function Type SEs

Just as with form and report function types, we need a predictor here to estimate the magnitude of effort involved with the Data function. We use the more common term "data element" rather than screen field in this context. The data element can be defined as an attribute or a column belonging to a table. For example, in the relation ***student (student#, name, city)*** we have three data elements.

The **data element** predictor measures effort involved with various data centred activities of 4GT development, such as data definition, creating tables and views, and entering sample data into such tables. Such activities consume effort and are indeed a component of the application software delivered to the users. Results obtained with the 4GT Model indicate that the data element integrates satisfactorily in our model. Other researchers have also demonstrated that the data element is a useful and essential predictor for estimating information systems. For instance, Wrigley and Dexter's research model indicates that *fields in files* correlates very well with application size.[59] Symons also proves that data elements are a good measure of size in the components of his model.[38]

3.5.3.4 Process SEs

Using SEs or screen fields to predict effort involved with process functions is difficult due to the variations involved in coding. For example, one can write a menu program (responsible for transfer of control from one module to another) using a 3GL with several different logic, or even generate such code with menu generators such as ORACLE's SQL*MENU. (The latter will take significantly less effort than coding with a procedural or even a non-procedural language.) However, it is beyond the scope of the present 4GT modelling research effort to identify all valid process SEs. Nevertheless, it remains an important topic and is being researched at present.[59]

3.5.4 Conclusion

In conclusion, we note that the predictor SE satisfies the constraints defined for a 4GT predictor — it supports both the specification oriented paradigm of 4GTs and the use of a screen field as a parameter. At present nine distinct categories have been identified — four for the form functions, four for the report functions, and one for the data function. Classifying SEs into the above categories serves the purpose of simplifying cost estimation using them. To illustrate with an example, the specifications "automatically retrieving name and address when ID is entered, and restricting data entry to one of four values are grouped in one category (detailed SE), whereas displaying screen field in upper case is grouped in another (Basic SE category). The former involves sophisticated specifications or use of macros (which is more time consuming) and the later involves simple specifications (i.e., straight forward menu selection key strokes).

Experimentation with the model, as described in Chapter 5, seems to indicate that the above categories of SEs are sufficient and satisfactory for modelling 4GT applications. Moreover, the above SE categories are substantially distinct from each other and therefore easy to apply them during cost estimation. This agrees with Boehm's views that a cost model must be simple if it is to be of any value at all.[16]

The SE as introduced here can also be viewed in terms of a *software brick*. Connell and Shafer's concept of a software brick is explained as follows:[60]

> If a brick wall is to be built, there are metrics available regarding the average amount of time required to lay one brick. Estimating the time required to build a wall then is reduced to simply calculating the number of bricks required from the wall's dimensions and multiplying that number by the current metric for brick laying.

With the 4GT Model, the total number of screen fields is equivalent to the "total number of bricks". The nine categories of SEs equate to nine different "sizes of bricks". Knowing the total number of screen fields in each category (i.e., total number of bricks required) and their corresponding metric values (i.e., time required to lay each brick), one can obtain an estimate of the total development effort!

3.6 A Model for Effort Estimation

The 4GT Model architecture is introduced in this section. Terminology, equations, design, mechanism, and use of the predictor SE for effort estimation are presented in this section. Model calibration, usage, installation and validation details appear in Chapter 5.

Two sets of factors influence the ultimate cost of a system

- Application Size Factors, and
- Project Factors

The *application size factors* are related to the *size* of the application and the *magnitude* of effort required to implement it. Overall application size is governed by the total number and type of functions identified for an application (as introduced in section 3.4, these are form, report, data and process functions). At the function level SEs govern the application development size (as explained in section 3.5.2).

The *project factors* are a group of project parameters that influence project costs. Skills, experience of the participants, methods or languages used, etc, are examples of project factors. These factors acknowledge the fact that application development effort and duration are affected by variables such as programmer skill, end-user skill, and familiarity with hardware. (Chapter 5 describes this aspect.)

3.6.1 Overview of the Model

The 4GT Model is essentially a *bottom-up* model as it uses specification elements as a means to estimate function size — collectively, the functions represent information system size.

A function can be classified into one of the following function types: form, report, process, and data. System models produced by the analyst during the systems analysis stage (such as data flow diagrams and data models) are a good starting point for identifying functions. Form and report functions are implemented using a form generator and a report generator respectively. Data functions are implemented using a DBMS.

Total Project Effort [E]
includes
- all life cycle activities
- project management
- team meetings
- documentation development
- prototyping overhead

Development Effort [D]
includes
- data definition
- coding & code generation
- unit testing

Figure 3.7: Conceptual View of Effort Distribution by 4GT Model

In order to further estimate the effort involved in implementing each function type, we use the predictor SE (introduced in section 3.5.2) for such purposes. Each SE has an effort value in person-hours associated with it; this is the work effort required to implement one SE (and hence one screen field). By using the techniques described in the model one can directly determine this effort value — no historical data is involved at this stage.

SEs requiring similar implementation effort are grouped together; this serves to reduce the total number of SEs. Nine such SE categories have been identified in the model — four each for form and report functions, and one for the data function.

Within each function, the total number of SEs multiplied by their respective effort values determines the *development effort* for that function. This mechanism therefore sizes a function. On summing the development effort for all form, report, and data functions (including process functions, if necessary), we have the total *development effort* (D) (see Figure 3.6). As the figure indicates, D does not include the complete life cycle effort or activities such as project management, developing user manuals, user interviews, administration, and team meetings. In order to obtain the *total project effort* (E), we multiply the development effort with an *expansion factor*. The expansion factor is derived from historical software metrics data of the organization installing the 4GT model.

Finally, *Project Factors* (PFs), such as "developer familiarity with 4GTs", "programmer skill", and "environmental factor" are evaluated for a proposed project on hand. An expert system has been implemented for adjusting project factors. It corrects the total project effort (E) to give us the final effort estimates for a project.

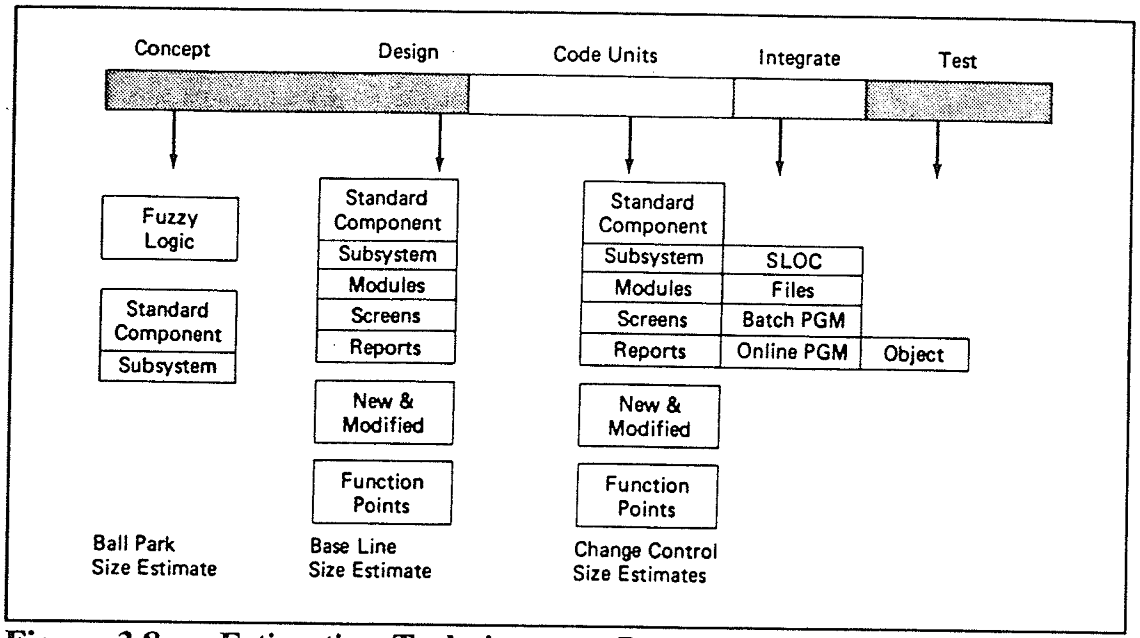


Figure 3.8: Estimating Techniques — Putnam & Myers

Putnam & Myers describe size estimation as a continuing process (see Figure 3.7). The figure indicates that various size estimating techniques can be viewed to be "strung out along the time line".^{106} Three major techniques are identified by them for estimating purposes: ball-park, base-line, and change-control. With *ball-park* sizing only a rough assessment of size is possible, however, such information might enable the project manager to decide whether it is acceptable to pursue a project further or not. *Base-line* sizing is a suitable technique when some software design is complete — it provides better results than ball-park sizing. *Change-control* sizing refines size estimates during the coding stage. It can give very accurate results for sizing the growth in code and for sizing the software integration phase.

In terms of the above framework the 4GT Model supports *base-line sizing* and *change-control sizing* fully. This is due to the ability of the model to size reports and screen modules satisfactorily. Attempts have also been made to use the 4GT Model for *ball-park* sizing (by using the average function effort as a basis — screen fields and SEs are obviously ignored as preliminary design has not occurred yet; see Chapter 6 for details).

Table 3.5: Concepts for Discerning Requirements

Abstraction	Suppress the details and concentrate on the essential properties of the system under consideration.
Partition	Represent the whole system as the sum of its component parts.
Projection	Represent the system using only a subset of its properties.

In conclusion we indicate that the 4GT Model architecture incorporates fundamental concepts such as *abstraction, partition*, and *projection* (see Table 5.1).[38] For instance, a) the model focuses primarily on the essential characteristics of a system being estimated - namely its functions; b) the system is partitioned into functions; functions can be summed up to represent the whole system; and c) SEs can be used to project effort related to development of functions

3.6.2 Deriving the Total Effort

This section derives the total effort (E) for the 4GT Model. It proceeds as follows: first we derive the basic development effort (D) required to implement all program functions, then we derive the total system development effort (E), and finally we illustrate how E can be adjusted for project factors (E adj).

Determining the Basic Development Effort

So far the following aspects of the model have been characterized:

- an application can be broken down into several functions
- each function can be classified into function-types
- each function-type has several Specification Elements (SEs) each of which can be classified into a category.
- each SE category has an associated SE Value (SEV). The SEV indicates the amount of time (in person-hours) it would take to

implement an SE. This information is stored in a Specification Element Table (SET)

Notation:

The set S consists of four function types

S = {form, report, data, process}

Each of the above function types has a Specification Element Table (SET), where the effort values for each SE category are stored; these are SET(form), SET(report), SET(data), SET(process)

In a SET(type),

- SE(type, k) is the kth SE category and SEV(type, k) is the associated SEV
- N(T) = number of SEs in the function type T

A project has n functions, denoted by F(i), i = 1,2,3,..,n, where

$F(i) \in S$

For the ith function F(i), let T = type of F(i). Considering SET(T)

- for k = 1 to N(T)

  set Count(i,k) to the number of SEs that falls into the category SE(T,k).

$$TSEV(i) = \sum_{for\ k=1}^{N(T)} Count(i,k) \cdot SEV(T,k) \tag{3.1}$$

We now have the basic effort D required to develop all program functions as,

$$D = \sum_{i=1}^{n} TSEV(i) \tag{3.2}$$

Determining the Total System Development Effort

The effort basic effort D derived in Equation 3.2 does not include the full *life cycle effort* but only the effort involved in the development stage. Effort due to the life cycle stages requirements definition, detailed design, etc., as well as activities such as project management, administration, meetings, and documentation development are not included in D.

Researchers and practitioners have successfully used expansion ratios for determining the total effort involved with a project.[13,16,97] For instance, Grady and Caswell, using their historic metrics database of several projects successfully demonstrate that elapsed phase effort can be used as a basis to estimate size (pp.140-147).[13]

Such a strategy is used by the 4GT Model; effort pertaining to a phase (i.e., development) is used as a basis for distributing effort across the life cycle (see Figure 3.6). This approach is similar to the "component ratios" described by Putnam and Myers for cost estimation purposes,[106] the difference being that the development component size is multiplied by a ratio reflecting the true overhead due to an organization's SDLC. We have validated this strategy of the 4GT Model — Chapter 5 illustrates this aspect in detail.

To obtain the total system development effort ***E***, we multiply the effort D due to basic development with an *expansion factor* reflecting the size of the remaining life cycle activities. The total system development effort is given by the following equation:

$$E = D * Expansion\_Factor \tag{3.3}$$

Determining the Expansion Factor

The expansion factor itself is determined from historic data by dividing an organization's *actual system development effort* with the *basic estimated effort* D for a project. The reason being that while the actual system development effort includes the effort due to project management and administration, etc., the effort

E does not do so. Therefore, the ratio between these two variables will give us an expansion factor that represents effort due to the entire life cycle.

$$\textit{Expansion Factor} = \textit{Actual System Development Effort} \,/\, D$$

As every organization uses a different methodology and also practices project management differently the expansion factor values may differ from one location to another.

Determining the Adjusted Total System Development Effort

Finally, Project Factors (PFs), such as "developer familiarity with 4GTs", "programmer skill", and "environmental factor" are evaluated for a proposed project on hand. Such factors influence the ultimate cost of a system and cannot therefore be ignored. An expert system called PFES has been implemented for making such corrections. It adjusts E above to give us the adjusted effort estimates for a project (E Adj.).

$$E\ Adj. = E * Project\_Factor\_Correction \qquad \textbf{(3.4)}$$

Chapter 5 covers project factor evaluation and correction in detail.

3.7 Using the 4GT Model

To use the 4GT Model for predicting effort requires the following steps:

1. The project planner uses all available information related to the project to decompose the software into functions — form, report, data, and process (if relevant).
2. The screen fields involved with each of the above functions are classified into SE categories. (This information can be summarized in the 4GT Model template illustrated in Table 3.6).
3. The 4GT Model equation is invoked at this point. This determines the development effort. (Steps 3, 4, and 5 are summarized in the template illustrated in Figure 3.7.)
4. The organizational expansion factor statistics is entered into the template. This determines the total system development effort.
5. Projects factors are evaluated at this stage, if necessary — and this gives us the adjusted effort.

Details pertaining to calibration of the model, including the recommended procedure for doing so, are described in Chapter 5.

Table 3.6: Template for Summarizing Data Related to Functions

4GT Model - Template 1:			
SE Category	SE Value (for Oracle)	Magnitude (No. of SEs)	Total Specification Effort (SEV * Magnitude)
F O R M S			
Simple SE	0.13		
Basic SE	0.29		
Detailed SE	1.59		
User Exit	22.73		
Total Form Effort			
R E P O R T S			
Simple SE	0.13		
Basic SE	0.84		
Detailed SE	2.55		
Total Report Effort			
D A T A			
Data Element	0.41		
Total Data Effort			

Table 3.7: Template for Related to Effort Estimation

4GT MODEL - Template 2.	
Form (from Template 1):	___ person-hours
Report (from Template 1):	___ person-hours
Data (from Template 1):	___ person-hours
Total 4GT Development Effort:	___ person-hours
Organizational Expansion Factor:	3.10 (Great-West Life)
Project Factors Correction:	___
Adjusted System Development Effort:	___ person-hours

3.8 Conclusion

In this chapter we introduced the life cycle associated with 4GT development and described how we can estimate information system size using functions such as form, report and data. Effort involved with each of these functions can in turn be predicted using a predictor which we call "specification-element". Just as 3GL cost models measure the size of an application in terms of LOCs, the 4GT Model attempts to do so in terms of the "specification elements" Considering that specification operations substitute physical coding with 4GTs, and that it is difficult, if not impossible, to collect project statistics in terms of LOCs — the techniques suggested by our model are valuable for cost estimating in the fourth generation domain.

Chapter 4

Evaluation of Project Factors

4.1 Introduction

In this chapter we examine various factors that affect the cost of software projects. These factors are referred to as *project factors* (PFs) in the context of the 4GT Model. They acknowledge the fact that the cost of developing an application is affected by variables such as programmer skill, end-user skill, or familiarity with hardware and software. A prototype model called PFES was implemented to model PF correction. It is fully functional at present.

The objective of this chapter is primarily to:

- identify and document all PFs that affect effort estimation in the 4GT environment
- to provide a mechanism for correcting the raw estimates generated by the 4GT Model
- to demonstrate the role that knowledge-based systems can play in cost estimation, especially when end-user computing is involved

The 4GT Model (as introduced in chapter 3) is not sufficient enough to explain the variation of software project costs. Therefore, various PFs that influence the development costs are evaluated here. It results in refining the effort estimate (E). Such PF corrections are applied in every instance of the model usage. Two basic schemes can be considered for applying PF corrections.

(i) Conventional Approach

(ii) Knowledge-Based Approach

Models such as COCOMO and FPA use the conventional approach for performing PF correction. We use the knowledge-based approach in this thesis as very little research has been conducted in the domain of "knowledge-based software cost estimation" — most of the existing cost models use the conventional approach only. We now have an opportunity to explore some of the presumed benefits such as — explanation capability of knowledge-based systems, ability to integrate project management experience (on completion of a project), and ability to train novice project managers or cost estimators. But first we review the conventional approach.

4.2 Conventional Approach

To illustrate the conventional approach we describe how COCOMO and FPA perform PF correction.

4.2.1 PF Correction in COCOMO

The various factors involved in COCOMO's PF correction were extensively researched and documented by Boehm.[16] Based on literature research, and his project data, fifteen *factors* (called cost drivers) were identified — see Table 4.1.

Table 4.1: COCOMO Cost Driver Attributes

1 . **Product attributes**
   a. required software reliability
   b. size of application data base
   c. complexity of the product

2. **Hardware attributes**
   a. run-time performance constraints
   b. memory constraints
   c. volatility of the virtual machine environment
   d. required turnaround time

3. **Personnel attributes**
   a. analyst capability
   b. applications experience
   c. programmer capability
   d. virtual machine experience
   e. programming language experience

4. **Project attributes**
   a. modern programming practices
   b. use of software tools
   c. required development schedule

4.2.2 PF Correction in FPA

As indicated in Chapter 2, the FPA method[41] adjusts the *initial_FP* count by applying a set of project factors. The final FP count is obtained using the equation:

$$FP = Initial\_FP \times [0.65 + 0.01 \times SUM(PF_i)]$$

Here Initial_FP is the unadjusted FP count obtained by evaluating the business functions. PF_i (i = 1 to 14) are FPA's project factors, also called *complexity adjustment values*, and are based on answers to the following questions:[42]

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?

12. Are conversion and installation included in the design?

13. Is the system designed for multiple installations in different organizations?

14. Is the application designed to facilitate change and ease of use by the user?

A value ranging from 0 to 5 can be assigned to each of the above PFs. This results in a correction range of 0.65 to 1.35 for the initial_FP count.

4.2.3 Summary

Similar conventional techniques are used by other researchers in their cost models. For example, Walston and Felix (see pages 54-73)[43] and Capers Jones (85-208)[44] describe similar factors. While some differences exist, one can notice some common grounds in each of the above models with regards to PFs used. For example, Siba Mohanty[45] compared fifteen conventional models used in various estimating methods by studying 49 PFs. The overall result of his comparison reveals that the following three project attributes are found most frequently, and account for 75% of the total attributes found in the various models,

size	20%
complexity	19%
environment	36%

It can therefore be assumed that when it comes to estimating software development, the three major concerns are size, complexity, and environment.[46]

4.2.4 General Strengths and Weaknesses of the Conventional Approach

This section reviews the conventional approach for its strengths and weaknesses. First the strengths:

1. The conventional approach is concise. It is easy to design and implement such a scheme since basically only a few key project factors, such as project environment and personnel ability are used.
2. It is more objective than the knowledge-based approach, and there is less room for influencing the ultimate estimates.[16]
3. Most such PFs are also objectively calibrated to previous data.
4. It is conceivable that such an approach can also outperform expert judgement due to the *bootstrapping phenomenon.*

The weaknesses of the conventional approach basically complement the strengths of the knowledge-based approach. According to Boehm, with the expert judgement approach (he was referring to human experts, not knowledge-based systems) there is an opportunity to factor in exceptional personnel characteristics or interactions, and other unique project considerations as well. This open-ended scheme is desirable, as indicated by Symons, when commenting on FPA's limited number of technical complexity factors. He states, "the restriction to 14 factors seems unlikely to be satisfactory for all time. Other factors may be suggested now, and others will surely arise in the future."[38]

One way to address the above problem is to use the knowledge-based approach. Several user friendly shells are available today and they readily assist us with encoding project experience and knowledge. The knowledge-based approach was a very suitable research method as it supplemented the sparse literature information available on the topics of end-user development and 4GTs. By contrast it was easy to find practitioners who had significant experience in this area and could readily qualify as "experts". Our basic purpose then became one of documenting such unrecorded expertise. It is a sad but true reflection of the state of software development — practitioners with knowledge do not have the time or inclination to publish.

4.3 Knowledge Based Approach

The knowledge-based approach can be defined as providing an explicit and visible representation of the knowledge. Even though there is an inferencing mechanism it is hidden away from the knowledge. In comparison in a 3GL program the same knowledge (called data) is embedded within the procedural code.

The knowledge-based approach provides us a more open-ended approach for effort correction. For example, consider the PF pertaining to end-user computing. COCOMO and FPA ignore this parameter, but with an open-ended PF correction scheme project managers or estimators can integrate the above PF into a knowledge-base by describing all relevant rules to the system.

Another advantage with the knowledge-based approach is that it provides us with the opportunity to model situations or attributes for which we do not have adequate statistical information but have heuristics or rule-of-thumb's. To illustrate with an example, very little information is available in the general literature today with regards to the impact of end-user computing on cost estimates. However, it is possible to resolve this difficulty if a project manager's experience can be represented in a knowledge-based system.

Other desirable side-effects of using the knowledge-based approach is that

(a) it can provide a very user friendly interface for requesting information from the users about various aspects of their application or the project environment (refer to Appendix A, illustrating the PFES interface), and

(b) it can offer a "consulting opinion" via the use of "How" and "Why" features of such a system.

(c) it can train novice project managers or cost estimators.

(d) it can provide a "what if analysis" capability.

Literature review reveals that at present no such specific system exists for 4GL cost estimation or end-user computing. Exploring models and estimation techniques using knowledge-based applications is a worthwhile goal today as expert systems technology has matured. Due to the availability of *expert system shells*, it is possible to easily integrate the expertise of the estimator, project manager, or any development team member into a knowledge-base today. (Little or no traditional programming is involved here; and a programmer is no longer needed to maintain or implement simple expert systems).[47] The commonly acknowledged knowledge-based benefits are also presented in Waterman[27](pp. 6-7). We can summarize the strengths as:

- an explanation capability
- an ability to provide *expert judgement*
- a basis for novice staff to perform project estimates
- a mechanism to provide a solution even if no algorithm is known, data are incomplete, or data do not exist (e.g., if historical metrics data is unavailable)

In describing the advantages of using the expert judgement approach for cost modelling Boehm states, "an expert's judgement is able to factor in the differences between past project experiences and the new techniques, architectures, or applications involved in the future projects. The expert can also factor in exceptional personnel characteristics and interactions, or other unique project considerations"(p. 333).[16]

4.4 PFES: An Expert System for PF Correction

In this section an expert system for correcting the initial software development effort estimates, ***Project Factors Expert System (PFES)***, is presented. The prototype PFES is designed to provide PF correction when end-user computing is used as well. PFES has been designed to function with the 4GT Model, but it can readily be used with other models as it simply presents an average correction factor. The test results from PFES execution have been found to be quite adequate and interesting as well. Since the PF correction issue can be resolved with PFES quite naturally by manipulating symbols and symbolic structures, it is inferred that knowledge-based application development is feasible and possibly suitable.

4.4.1 Development Methodology

The life cycle methodology for developing knowledge-based systems is different when compared with traditional systems. The following stages were used to implement PFES.[48]

(a) Identification

This is the first step in Expert System building development, and the objective here is to: first define the problem, and then to describe the important features of the problem such as type and scope, goals and objectives, participants and resources. It also includes a discussion of several topics such as current system (if any), proposed system, scope of the system, participants, alternatives and goals.

(b) Conceptualization

During this stage the concepts, relations, and control mechanisms needed to describe problem solving in the domain are presented. This is a critical phase — successful extraction of major and minor concepts can alleviate hardships at latter stages. The conceptualization task for PFES falls somewhere in between easy and hard, and a significant proportion of time was spent during this phase.

(c) Formalization

Waterman defines this stage as follows: "Formalization involves expressing the key concepts and relations in some formal way usually within a framework suggested by an expert system building language." Matching the application characteristics with the appropriate tools is one of the key concerns here.

(d) Implementation

During this stage we convert the formalized knowledge into a working knowledge-based computer program. If an *expert system shell* is available, and matches the application characteristics it can be used. Otherwise a new tool has to be developed using A.I. languages such as LISP or PROLOG.

(e) Testing

This is the last stage and we evaluate the performance of the prototype here. Various test cases representing as many scenarios as possible are analyzed by the knowledge-based system.

4.4.2 Overview of the Prototype Development

The scope, participants, computing tools, validation strategy, and other aspects related to the PFES prototype development are presented here.

Problem Definition

A large number of factors can affect the software development estimates. The primary objective of PFES is to recommend PF correction for small to medium sized 4GT software development projects. Large application systems are ignored since the 4GT Model is not concerned with larger projects. It provides estimates when either end-user computing, or traditional DP department based development is involved.

Scope

The scope of PFES is to a certain degree constrained by the scope of the 4GT Model — effort correction of small to medium business applications using 4GTs. The overall scope is narrow enough to make the problem manageable and sufficiently broad at the same time to ensure that the problem has some practical interest.

Participants

Norbert Kaehler, Assistant Manager, Development Services, Information Systems and Data Processing, Investors Group, Winnipeg, functioned as the *expert.*[77] He suggested various project factors that are relevant to application development in the fourth generation environment, but do not exist in current literature. He also evaluated the relative importance of other PFs in light of 4GT development. Other ISDP staff at Investors were also consulted as required. The nature of consultation ranged from collecting actual historic data pertaining to implementation of 4GL and 4GT projects, to seeking a second opinion from them with regards to the validity of any project factor (based on their experience as project managers). The knowledge-engineering and programming was performed by the author of this thesis. Kaehler subsequently validated the model using several case studies from his experience. Members of the examination committee, especially Scuse, contributed throughout the knowledge engineering process and eventually by testing the PFES prototype.

Knowledge Acquisition

The domain of knowledge required for PFES was obtained from the expert, but literature search, consultation with other experienced project managers, and the author's personal experience with project management also played a useful role.

Computing Tools

VP-Expert, an expert system shell from Paperback Software International, was used to implement PFES. It is a rule-based system for developing small to medium sized expert systems and is suitable for developing initial prototypes, as in our case. It runs on a IBM PC microcomputer or other PC compatibles. It is capable of using both forward and backward search when searching for a solution.

It is also capable of interfacing with databases created using Ashton Tate's dBASE III and IV packages, and spreadsheet's created with LOTUS 1-2-3.

Validation

Various test cases representing typical variations in developer ability, project environment, and product complexity were generated and tested using the prototype system. Validation process involved expert approval of both the individual PF values embedded in the rules, such as reliability, interface complexity, etc., and the overall PF correction value suggested by PFES.

4.4.3 Overview of the Development Process

The following steps provide an overview of the development process used for system development:

(1) identify the knowledge acquisition method

(2) identify all project factors that can affect the cost of development in the 4GT environment

(3) group them into categories that affect estimates, either positively or negatively

(4) determine the range of values (effect) for each of these groups

(5) design the conceptual model, implement an initial prototype, and execute it against test data

(6) validate the output and redesign the model if necessary

(7) present the recommended PF correction.

4.4.4 Conceptual Model

The various PFs that play a central role in PFES are described here. Albrecht,[18,41] Watson & Felix, DeMarco,[61] Boehm,[16] Capers Jones,[44] Basili,[43] amongst other researchers have identified and used some of these PFs in their cost models. However, our knowledge engineering process succeeded in revealing some new PFs for our problem domain.

The different PFs that play a role in the knowledge-based system are described here next. The first PF (mode of development) relates to end-user computing. This is a new PF and does not exist in other models.

Mode of Development

It is possible for end-users today to tackle challenging data processing tasks, and they indeed are doing so in several organizations. When end-user based application development is taking place it is necessary to measure two concerns:

a) the extent of support available from various sources such as Information Centre, or assistance with various aspects of application development is available from the data processing shop. If such support exists then it serves the purpose of facilitating application development and eventually reducing effort and cost of software development.

b) Six categories of end-users have been identified by Rockart and Flannery.[49]

   (a) Data Processing Programmers

   (b) End-user Computing Support Personnel

   (c) Functional Support Personnel (power users who work in functional departments, outside of IS)

   (d) End-user Programmers (who can write code)

   (e) Command Level End-users

   (f) Non-Programming End-users

Basically, we classify the above into three general categories — at the lowest end of the spectrum we have *application users* who are not capable of any end-user programming or even executing simple commands, but can operate a menu driven application system, and at the upper end we have *programmer end-users* who are basically full time system developers, highly skilled, and well qualified. The middle category ranges from end-users capable of some command level operations (and therefore capable of 4GT programming with adequate training or support), to sophisticated dBASE programming types, but less sophisticated than the professional programmers.

For the purposes of PF correction, end-users at the lowest end and the highest end of the spectrum are ignored. The *non-programming end-users* probably will never be requested to develop 4GT applications, and the *programmer end-user* can very well be classified in the same category as the professional data processing staff member.

Previous Familiarity

Previous familiarity with the application environment identifies attributes such as:

— experience with similar size of applications

— experience in the application problem domain (e.g., accounting applications)

— previous experience with similar hardware and operating system

— previous project team experience

Individual capabilities are considered here for project factors such as the ones listed above and described in terms of "years of experience." If the end-users are involved with development, only *full time equivalents* of "years of experience" should be considered. If the participants are very experienced it can reduce the development effort significantly.

Previous experience with 4GT tool

This project factor quantifies previous experience with 4GTs and with the DBMS. Specifically, it is concerned with the extent of experience with the specific 4GL and DBMS to be used in application development either by the end-user or DP staff.

Project Novelty

This attribute was identified as the single most significant factor by Crossman[117] in his investigation of application development productivity. His analysis of data revealed that novel applications took up significantly more resources (effort) than familiar ones.

Application Factors

Application factors are concerned with the details of the current application being developed. They identify the complexity of attributes such as:

— reliability required of the new application

— data communication involved

— designing applications that facilitate change, or reuse of code

— interface complexity (I/O complexity due to printing on laser printers, or VGA monitor in colour)

— providing operational ease

Methodology Factors — Practice

Practice factors are concerned with the development rigour of the application development environment. It is concerned with the use of techniques such as:

— top down design

— structured systems analysis techniques

— formal walkthrough's

— acceptance testing

— use of structured programming techniques

— use of automated tools for flowcharting, documentation, testing, etc.

Methodology Factors — Techniques

The technique factors are concerned with use of techniques such as JAD and Prototyping in the methodology. The expert has indicated that such factors can affect software estimates significantly.

When end-users are developing systems the option "PF not applicable" may be selected. In those cases where the end-user itself is the user, i.e., JAD and Prototyping is meaningless then.

Other Factors

This provides us the opportunity to consider other PFs that might affect the overall PF value. They include:

— Staff morale

— Staff Compensation

— Xeroxing & Printing resources

— Individual Workstations

— Technical Education

— Availability of essential Software & Hardware

— Working on a Low Priority Project

— Staff Working on Several Projects Concurrently

Attributes such as "Travel Involved" may be incorporated here if so desired. But care should be taken to see that this is not done indiscriminately. Note that some of the PFs described in this section correlate positively to effort, and some negatively, and also some PFs have a more significant impact on the effort estimates than the other. Both these aspects are dealt with by the cost model.

4.4.5 Calculating the Effect of Each PF

There are several options available to determine the effect of each PF and also to calculate the overall PF correction. Simple addition of the PFs, or multiplication of the PFs, or even subjective techniques can be used to determine the effect of PFs. The strategy used here is a heuristic one and is comparable to that in COCOMO. Boehm uses a heuristic approach to determine the effect of each PF; he first quantifies them by assigning ratings to each attribute on a five- or six point scale (such as Very Low, Low, Extra High), and then allocates numerical values to the ratings (such as 1.15 or 0.75).

The following heuristic is used in PFES to determine the effect of each PF:

1. Assuming that PFs are normally distributed in the population, the *normal curve* can be used to approximate the PF ratings and determine individual PF values for each rating. In order to do that we define each project factor as having a mean of 1.0, and a range of values X_1, X_2, or X_3 referring to significant-, average-, or moderate influence on estimates. One important characteristic of this approach is that all curves have the *mean* at the point where exactly half the population is below it and half above it, that is, for X_1, X_2, or X_3 what is true on one side of the curve is also true on the other side. The attribute

ratings, very low, low, average, high, very high, then can be defined symmetrically around the average as low/high, or very low/very high.

2. Initial ranges for X_1, X_2, or X_3 were based on literature review — primarily COCOMO, and were determined to be 0.14, 0.7, 0.35 respectively. In course of the knowledge engineering and validation process some of these values for were adjusted on recommendation from the expert. See Tables 4.2, and 4.3.

Table 4.2: Factor Values of Ratings Influencing Positively

Degree of influence	**V. Low**	**Low**	**Average**	**High**	**V. High**
Significant (x1)	1.28	1.14	1.0	0.86	0.72
Average (x2)	1.14	1.07	1.0	0.93	0.86
Moderate (x3)	1.07	1.04	1.0	0.96	0.93

Table 4.3: Factor Values of Ratings Influencing Negatively

Degree of influence	**V. Low**	**Low**	**Average**	**High**	**V. High**
Significant (x1)	0.72	0.86	1.0	1.14	1.28
Average (x2)	0.86	0.93	1.0	1.07	1.14
Moderate(x3)	0.93	0.96	1.0	1.04	1.07

3. During calibration some of the PFs were defined as being significant, average or moderate. For example, PFES defines novelty, experience

with 4GT tool, and overall experience, amongst others, as being significant. While these attributes are "soft coded" (see Appendix A, lines 1, 2 and 3), others are "hard coded" in the rule-base (as is done in the case of rules pertaining to reliability).

4. If a new factor is required to be part of the PFES this information is appended to the expert system, but until it is validated it should be classified as having moderate influence, therefore, assuming a range in the PF value of plus/minus 0.07, at most. In addition, the option "PF_not_applicable" should be provided, therefore making it possible to exclude this PF when appropriate.

4.4.6 Exceptions: Rating Values for End-users

In PFES some exceptions were made when evaluating end-user performance using the rating values of the above heuristic. By using expert opinion as a validation mechanism the following unique ratings were decided upon when end-user computing occurs:

Functional Support Personnel

V. Low	Low	Average	High	V. High
1.84	1.63	1.42	1.21	1.00

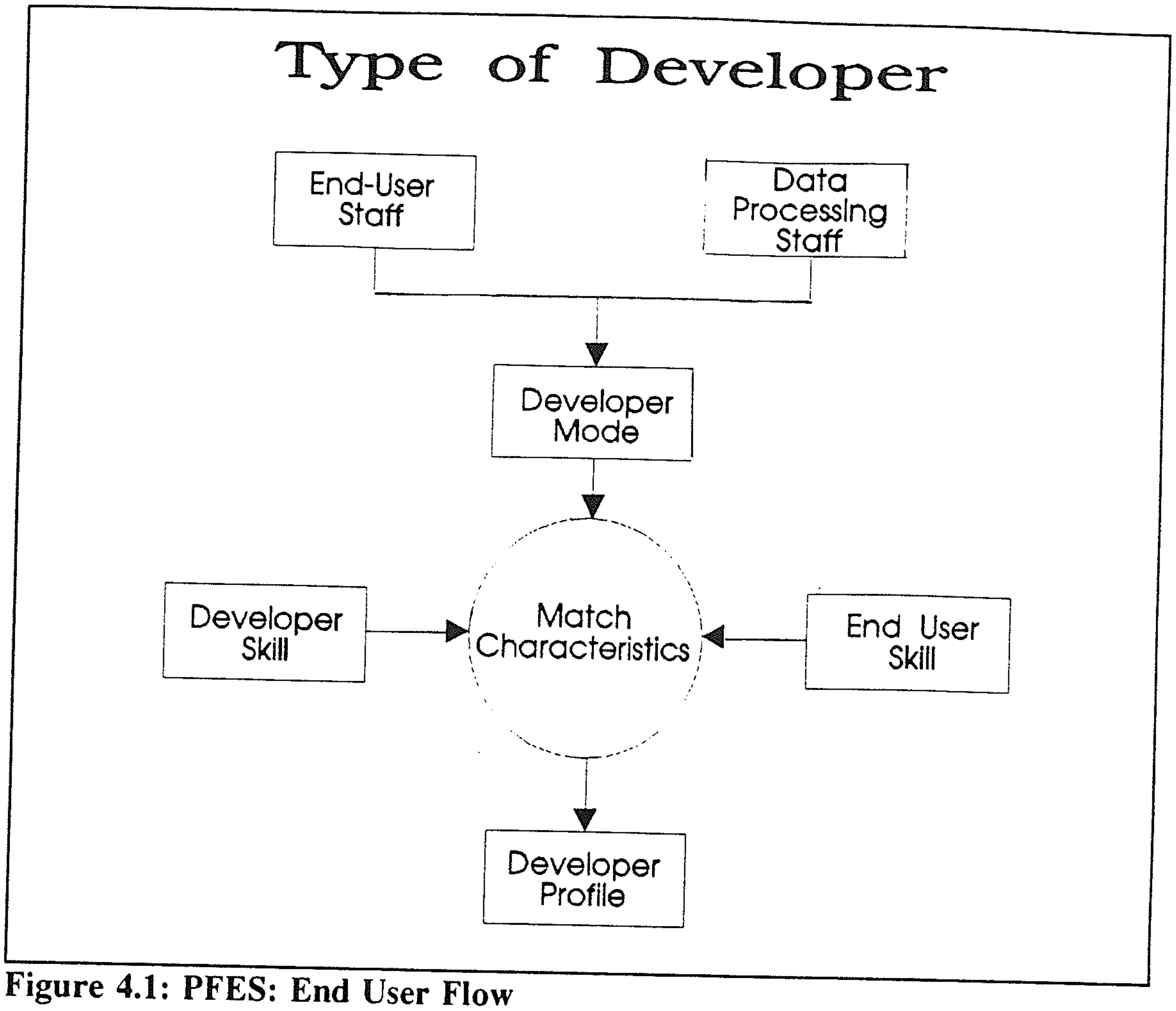


Figure 4.1: PFES: End User Flow

Since functional support personnel are experienced users they are considered as having an *impact* ranging from 1.0, (for very High Skill) to 1.84 (very low).

Command Level

V. Low	Low	Average	High	V. High
2.47	2.26	2.05	1.84	1.63

The command level end-users ratings average is three magnitude lower than those of the functional support personnel as the end-users is not skilled with most of the aspects of application development tools. Note that the above numbers have not been extensively validated in other environments and caution must be taken in using them elsewhere.

4.4.7 Validation of the PF Values

Boehm has suggested the following method to determine the effect of a PF value (p. 378).[16] This type of validation can be performed on completion of a project.

1. Using the 4GT Model compute the estimated development effort for a project without the influence of the effort multiplier PF being analyzed. Let this be called Effort_PF.
2. Define the ideal effort multiplier (IEM_PF) for this project. One which if used in the 4GT Model would make the estimated development effort for the project equal to its actual development effort (actual_effort). That is,

IEM_PF = Actual_Effort / Effort_PF

As an example, if Actual_Effort is equal to 1020 person hours, and Effort_PF (experience with 4GT tool) is equal to 1200 (using the above equation), the IEM_PF value is:

$$\begin{aligned} \text{IEM\_PF} &= 1020 / 1200 \\ &= 0.85 \end{aligned}$$

Note: Effort_PF is obtained by ignoring the impact of the PF representing the previous experience with 4GTs, assuming that is rated as "high," and influence being "significant."

Now, if this IEM value is compared with the one used in PFES, it must be reasonably close to 0.85. Such a procedure therefore serves to validate the various PF values.

4.4.8 Design of the User Interface

Data and information required by PFES to make a decision is obtained through user responses to questions. Some information, however, is obtained indirectly from the *rules* through inference. Actually, PFES requests data from the user only if it cannot resolve the goal or sub-goals internally using its rules. A listing of the questions asked by PFES is illustrated in Appendix B. The following ratings described the range of most of the attributes defined above:

very low, low, average, high, very high, PF not applicable

If it was determined that the option "PF not applicable" is not applicable for a given PF then this option is omitted. Finally we note that the questions used in PFES could not be more descriptive due to the limitations of the expert system shell.

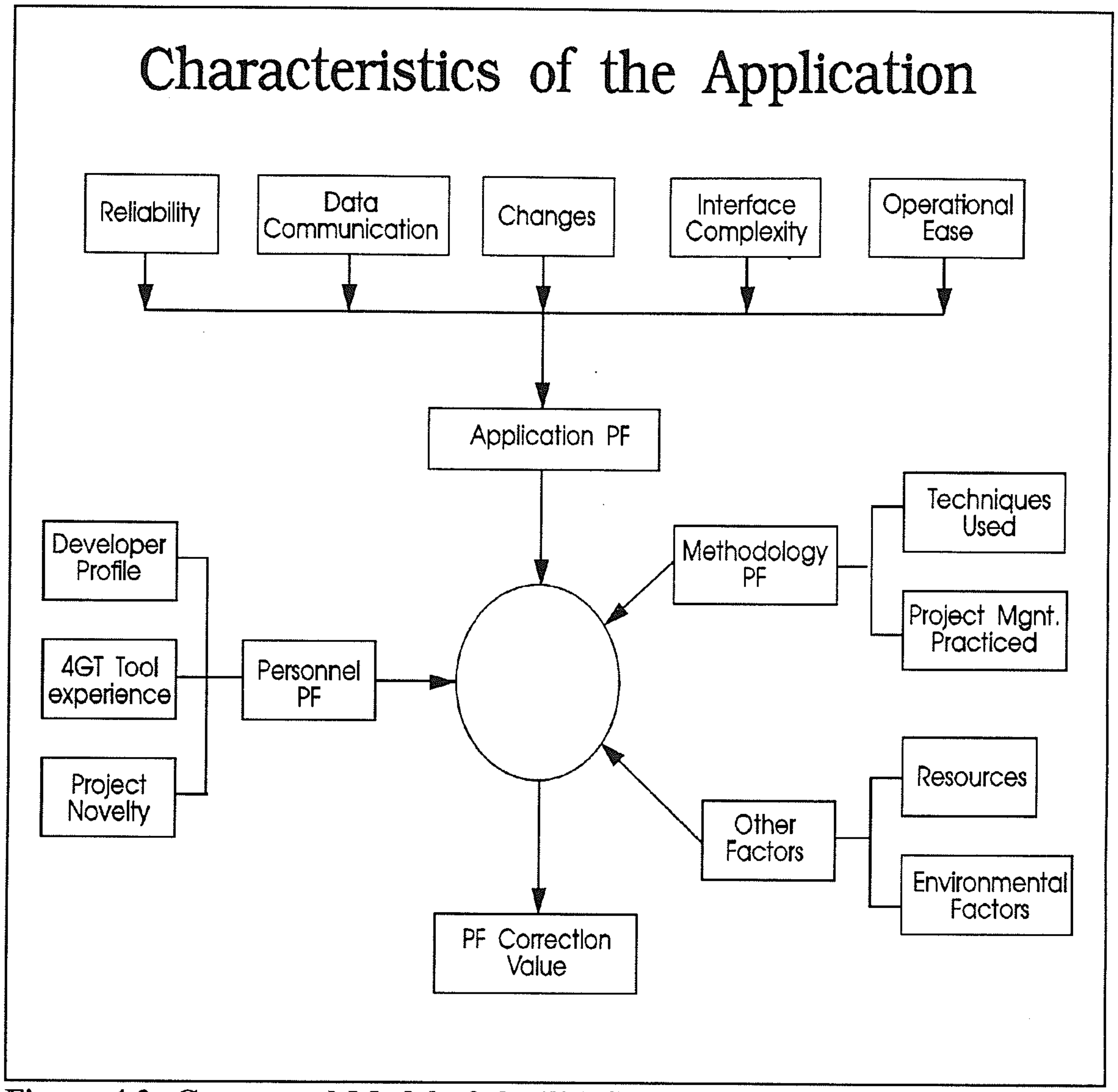


Figure 4.2: Conceptual Model of the PFES

4.4.9 Design of the Prototype

The concepts surrounding the application design are illustrated in the Figure 4.2. The detailed flow of reasoning and assumptions are not presented as it is too technical and prototype-specific, but generally speaking, this is what happens — PFES obtains the values of the individual PFs from the estimator, aggregates the values, and gives an overall PF correction factor.

4.4.10 Validations, Analysis, and Test Results

As indicated earlier, two stages of validations are attempted. First, the individual rules and their range of ratings are approved, and also matched with comparable PF ratings in literature (when available). Second, the PF correction value (output) from PFES is validated against expert opinion.

Note that adjustment and tuning of the prototype rules (and values) were necessary and were performed several times before satisfactory results were obtained. The following procedure was used — input information for the Personnel_PF was provided to PFES, and the results were validated for both different types of developers, the data processing staff, and the end-user. Different types of end-users with varying degrees of skills were verified. This was followed by independently testing the Application_PF and the various complexity attributes therein. Finally, the Methodology and Environment PFs were validated independently and their results approved. All of the above components were tested collectively, and for a range of input values the PF correction factor was validated against expert opinion.

The test results of three typical test cases found in the software development environment are presented, and three abnormal cases are presented below. The abnormal cases reflect two extreme cases, i.e., selection of all PFs that affect PFES negatively, and positively. The other case is where all average values are selected. The scenario where strong experienced personnel are given the most difficult projects and the weakest team personnel given an easy project are also depicted.

Results: Sample simulations from the initial version of PFES resulted in the following values. The results show some normal and extreme PF correction values.

Inexperienced Team Assigned Very Easy Project:

The 4GT tool value is 1.28

The Development teams familiarity with the application on hand is 1.28

The Personnel PF correction is 2.32

The Methodology correction factor is 1.0

The Project administration environment rating is 1.07

The Work and Staff environment rating is 1.07

The PF Correction is 0.51

(VP-Expert's execution trace for this result can be found in the Appendix E.)

Moderate Project Assigned to Capable Team:

The Overall experience value is 0.58

The 4GT tool value is 0.72

The Development teams familiarity with the application on hand is 0.72

The Personnel PF correction is 0.30

The Methodology correction factor is 1.0

The Application project factor correction is 3.43

The Project administration environment rating is 1.07

The Work and Staff environment rating is 1.07

The PF Correction is 1.18

Very Easy Project Assigned to Very Capable Team (extreme case)

The Overall experience value is 0.58

The 4GT tool value is 0.72

The Development teams familiarity with the application on hand is 0.72

The Personnel PF correction is 0.30

The Methodology correction factor is 1.0

The Application project factor correction is 0.19

The Project administration environment rating is 1.00

The Work and Staff environment rating is 1.00

The PF Correction is 0.25 (Case corrected by PFES).

Additional PFES Execution Results

Very Difficult Project Assigned to Very poor team

The Overall experience value is 1.42

The 4GT tool value is 1.28

The Development teams familiarity with the application on hand is 1.28

The Personnel PF correction is 2.326528

The Methodology correction factor is 1.0

The Application project factor correction is 3.435974

The Project administration environment rating is 1.00

The Work and Staff environment rating is 1.00

The PF Correction is 7.993890 (case not recommended by PFES).

Average Project Complexity and Average Team

The Overall experience value is 1.0

The 4GT tool value is 1.00

The Development teams familiarity with the application on hand is 1.00

The Personnel PF correction is 1.

The Methodology correction factor is 1.0

The Application project factor correction is 1.

The Project administration environment rating is 1.00

The Work and Staff environment rating is 1.00

The PF Correction is 1

Very Capable End-user Assigned Easy Project

The End-user value is 2.05

The 4GT tool value is 1.00

The Development teams familiarity with the application on hand is 1.14

The Personnel PF correction is 2.3370000

The Methodology correction factor is 1.0

The Application project factor correction is 0.231115

The Project administration environment rating is 1.00

The Work and Staff environment rating is 1.00

The PF Correction is 0.540116

4.5 Conventional PF Correction

If an organization is unable to use expert system technology for effort correction (due to lack of resources), it is still possible for them to use the 4GT Model (introduced in Chapter 3) in conjunction with a spreadsheet version of the PFES model illustrated in Appendix C. This version approximates the mechanics of the knowledge-based system. Of course, in this case, the various advantages of the knowledge-based approach, as narrated in this chapter, are not valid any more.

4.6 Conclusions and Contributions

Using the knowledge-based systems approach to estimate various project parameters provides us several advantages. Even though the bulk of the research related to PFES development occurred only at one site we feel that these advantages are relevant to most organizations involved with fourth generation development. To summarise:

- it provides an opportunity to capture the knowledge of an expert cost estimator and document them. This is especially useful if "hard" documented results are not available in literature yet due to the newness of the technology (as with 4GTs). Practitioners generally have useful experience, but either for proprietary reasons or for lack of interest do not often publish such data promplty.

- novice project managers can use the user-friendly interface of PFES to specify project parameters and study their influence on the overall effort required for software development.
- the explanation facility provides an opportunity to study the impact of various factors such as "programmer skills" or "prior familiarity with hardware" on the cost of developing software. The project manager can use this information to minimize the cost of developing software by choosing the best approach.
- an open-ended approach is facilitated for knowledge encoding. For example, if in the future, the technology changes, or any other criteria affecting costs come into picture, they can be incorporated into PFES to obtain better estimates. Due to the accrual of such relevant knowledge into the knowledge-base it is possible for project estimation accuracy to improve.

Incorporation of new project factors into the knowledge-base, however, must be done carefully as it can result in an incorrect correction. In order to resolve serious potential problems Kaehler[77] has proposed that we put caps on the range of influence of PFES corrections at 0.25 to 5.0. Such a strategy is evident in FPA and related cost models.[18] The above factors lead us to conclude that it is very important to validate an expert system periodically after any changes are made to it. (The procedure described earlier on in the chapter can be used for such purposes.)

- two unexpected but useful results from our experimentation with the knowledge-based approach turned out to be:
  (1) A new data group set pertaining to end-user computing
  (2) Rules that recomend whether end-user development occur or not, or that it proceed cautiously. (Such inferences could be useful for the manager).

Chapter 5

Model Experimentation and Analysis

5.1 Introduction

In this chapter we provide details of 4GT Model calibration, experimentation, and validation. Detailed procedures for installing and using the model are also provided here.

Participants from three different sources were involved with the process of calibrating and validating model weights[78,81,82]:

- Great-West Life Corporation
- Pitblado & Hoskin
- University of Winnipeg

This diversity served the purpose of indicating if the model is portable across organizations. (Section 3.1 provides a detailed history of the model including participants and dates.)

5.2 Implementing 4GT Model

First we recall that our 4GT model, as described in Chapter 3, is able to support the following objectives:

- *Estimation of development effort:* This refers to estimation of the development time (D) required to implement all functions of the delivered software (such as forms and reports).
- *Estimation of total system development effort*: This refers to the estimation of the total system development effort (E). It includes effort due to life cycle, project management, administration, and documentation.

This chapter examines both the above issues in depth. It is organized as follows: First, we illustrate how the different weights associated with the model are determined for a given 4GT tool. Then we describe how the *expansion factor* is determined for a given organization. Finally, we describe how the model can be validated. The above constitute the main thrust of this chapter, but other issues such as portability and productivity measurement of tools are also discussed.

5.3 Model Calibration

Ted Janzen, Associate Manager Computer Systems, Great-West Life, was approached in May 1991 for purposes of experimenting with the 4GT Model. Ted Janzen's Computer Systems group was actively developing 4GT based application systems using ORACLE and they were therefore quite interested in calibration and experimentation with the 4GT Model.

The objective of the calibration procedure is two fold:

1) To obtain the life cycle *expansion factor* for Great West Life.
2) Calibrate the weights associated with each of the SE categories (Simple, Detailed, etc.).

The calibration process performed is described in detail in the next few sections. But first we present some details of the host site and interaction with its participants.

5.3.1 Host Site and Participants

Great-West Life is a large corporation with offices across Canada and the United States. It is headquartered in Winnipeg, and provides a wide range of insurance, retirement and investment products to about six million people. Technology is at the core of their business, and more than 400 systems professionals support their diverse needs in Canada. Great-West Life has very good computing resources. The following are some of the major hardware and software supported: Hardware — IBM 3090 and 286/386 PC's; Software — MVS/ESA, CICS, IMS DB/DC, DB2, TELON, PL/I, COBOL, C, ORACLE, ACCEL and several PC-based software packages.

At our very first meeting, a report describing the 4GT Model was given to the interviewees at Great-West Life (see Table 3.1). The initial meetings served the purpose of acquainting the participants with the 4GT. Information about their project management and cost estimation practices was also obtained. Subsequently more than twenty five meetings took place between May and October 1991 [77-84] each lasted more than one hour, but usually less than three hours.

The initial few meetings highlighted the difficulty of calibrating and using the 4GT Model as originally presented to them. However, largely due to the efforts of Smith and Janzen the 4GT Model was successfully adapted for experimentation. Since ORACLE[58] was used to experiment with the model it is described next.

5.3.2 ORACLE Tools

ORACLE Corporation's ORACLE is a popular relational database management system that supports SQL. The user interface and SQL language are compatible with both IBM's DB2 and SQL/DS. ORACLE comes with a complete set of fourth generation support tools such as a menu generator, report writer, forms generator, and data dictionary. ORACLE is designed for a multi-user environment. The ORACLE environment consists of the following components: a relational database management system, an active data dictionary, SQL query language, application generator, and report writer. A brief description of ORACLE's family of application development tools is presented below:

SQL*Plus	Provides direct interface to the ORACLE relational database system. Contains the full implementation of ANSI SQL. Used to create and manage tables.
SQL*Forms	It is an interactive forms generator. It provides access to SQL for those applications that require its use. Complex processing is facilitated via user exits to system macros and other 3GLS (from within the form). Once an application is designed and generated it can be used by the operator for querying, updating and adding data.
SQL*Report Writer	Used to generate report

SQL*Menu	Can be used to create a menu driven system by integrating different functions.

Their CASE system environment consists of the following components: CASE * Dictionary, CASE * Designer, CASE * Generator, and CASE * Method - all fairly standard tools. ORACLE runs on IBM mainframes, DEC, high-end microcomputers, and several other computing environments. It was first developed for the MVS, VM/CMS environment but is now available under UNIX and PC-DOS. There is considerable code portability as all versions of ORACLE are identical and include the full implementation of SQL. In addition, ORACLE's network software allows networking of microcomputers, minicomputers and mainframes and permits sharing of databases.

5.3.3 The LEGASY Project

The LEGASY (*LEGA*l *SY*stem) project was used to calibrate the 4GT Model at the Great-West Life. Subsequently another project, Telephone System, was used to validate the calibrated weights of the 4GT Model. We present project details of the LEGASY project here. The Law Department within the company was interested in a legal system that met the following requirements:

(1) automated litigation management: store information regarding issues

(2) automated calendar of events: keep track of scheduled events of each file.

(3) automated time tracking: record in-house counsel time for each file.

(4) implementing key word document search: locates document on the system which contains a specific word or a phrase.

Corporate Systems examined the above requirements in June 1990, with a view to implementing the system. Selected details of the proposed system are presented in Appendix E. The system took 2340 person hours to implement. The following staff were involved at various stages of the project: Janzen, Warkentin, Smith, Trainor and Buskens (see Table 3.1). It started on November 20, 1990 and took ten months to complete it fully. In May 1991, the process of calibrating the 4GT Model began. The LEGASY project was selected by Janzen (Project Manager) for calibration purposes as he determined it to be an average project involving average experience. The fact that current data useful for cost modelling purposes were easily available was also a key reason.

5.3.4 Procedure Used to Calibrate Weights

The people participating in the calibration process[78] met the following two criteria:

1) They had one to three years of development experience using ORACLE and its tools, and
2) They had a previous experience of one to three years with the ORACLE environment (i.e., computer, operating system, utility tools, and methodology).

These two criteria are crucial as the 4GT Model is designed to provide us average effort estimates only. (Note that project factor corrections can take place if so desired using PFES — as discussed in Chapter 4.)

The following steps were used to calibrate model weights — they can, and must preferably be performed during the development stage of an on-going project. In our case the calibration was done with the LEGASY project (see section 5.4 for details):

1) Classify the functions into either form, report, or data types.
2) Identify the screen fields that constitute each function.
3) For form or report functions:
    a) First, determine the effort associated with implementing a skeletal screen. Divide this number by the total number of screen fields in each form or each report. This gives the Simple SE value (SF).

b) Next, for applicable cases, categorize the above screen fields into one of the SE categories — basic, detailed and user exit.

c) Document the effort associated with implementing each SE. Average the effort by SE category — this gives us the Basic (BSE), Detailed (DSE), and User Exit (UE) values.

4) For data functions:

a) Determine the total number of data elements in a table.

b) Determine the total effort involved with table definition.

c) Divide the total effort by the total number of data elements — this gives us the calibrated value for data-element (DE).

5.3.5 Calibration Details Pertaining to LEGASY

We basically followed the above steps in detail. First we identified functions and classified them in one of the three function types — report, form, and data. The functions not falling into the above three categories belonged to the process type — these were modules coded using C programming language. The effort associated with these were documented separately. (Only one person, Buskens, was involved with all C coding and he gave us all effort statistics related to C programming.) Next SEs were identified for each of the above functions and these were classified into SE categories. (Note that the scope of the research did not include the process type so we did not determine or calibrate their SEs). The following strategy was used to locate SEs.

(a) Form type: Count total number of screen fields. Classify applicable screen fields into either: basic, detailed, or user exit SE categories.

(b) Report type: Count the total number of following report fields — report screen fields, report summaries (and related objects) into SE categories. No user exits were identified as ORACLE does not support exits to procedural language.

(c) Data type: Count the total number of data elements in the whole database.

Note that the above strategy, at this stage, does not involve the use of historic project data in the conventional sense as only actual development effort values relating to the various SEs are documented here. This approach for computing basic development effort is unique amongst related cost models. It consequently provides us with some advantages:

- Dependency on large quantities of historic project data pertaining to the same fourth generation tool for calibration purposes is minimized (and so is the risk associated with using them.[22,23]
- Ease of re-calibrating values for newer versions of 4GTs.
- Ease of re-calibrating values for new 4GTs.

Results

Appendix F and G document statistics related to all the functions involved with the calibration process at the Great-West Life. In total more than twenty eight form functions were involved, with the process resulting in identification of 185 SF's, 36 BSE's, 32 DSE's, and 11 UE's. These SEs were calibrated using person-hours as a measuring unit resulting in the following weights: SF = 0.13, BSE = 0.29; DSE = 1.59; and UE = 22.73. For the report type we identified 59 SF's, 14 BSE's, 69 DSE's, resulting in the following weights: SF = 0.13, BSE = 0.84; and DSE = 2.55. For the data function type we identified 238 data elements in all the LEGASY tables.

The calibrated weights are summarized below in Table 5.1.

Table 5.1: Calibrated Weights for the 4GT Model

Project Participant	Function Type	Simple [SF]	Basic [BSE]	Detailed [DSE]	User-Exit [UE]	Data Element [DE]
GWL	Form	0.13	0.29	1.59	22.73	N/A
GWL	Report	0.13	0.84	2.55	N/A	N/A
GWL	Data	N/A	N/A	N/A	N/A	0.41

With the determination of the above weights it is now possible to state the 4GT Model equation — for example, at the Great-West Life, the following 4GT Equation is valid for predicting ORACLE project development effort:

$$D = [(SF_{form}*0.13)+(BSE_{form}*0.29)+(DSE_{form}*1.59)+(UE_{form}*22.73)+$$
$$(SF_{report}*0.13)+(BSE_{report}*0.84)+(DE_{report}*2.55)+(DE_{data}*0.41)]$$

5.4 Determining the Expansion Factor

Note that the above 4GT Equation is unable to predict the entire life cycle effort. Effort due to the various life cycle stages as well as activities such as project management, administration, meetings, documentation development are not included in D. In order to obtain the total system development effort ***E***, we need to multiply the effort D by an *expansion factor.* (Chapter 3 explains the expansion factor in detail.)

The expansion factor is determined by dividing the *actual system development effort* with the *estimated effort due to forms, reports and processes.* For the LEGASY project (see table below) this factor is equal to 2340/755.41 = 3.10.

4GT MODEL - Calibrated For Legasy				
Forms			335.37	person-hours
Reports			197.46	person-hours
Data Type			97.58	person-hours
Process Type			125.00	person-hours
Development Effort (estimated)			755.41	person-hours
Development Effort (actual)			2340.00	person-hours
Expansion Factor (actual/estimated)			3.10	
F O R M S				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	185	24.05	
Basic SE	0.29	36	10.44	
Detailed SE	1.59	32	50.88	
User Exit	22.73	11	250.00	
Total Effort			335.37	person-hours
R E P O R T S				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	75	9.75	
Basic SE	0.84	14	11.76	
Detailed SE	2.55	69	175.95	
Total Effort			197.46	person-hours
D A T A				
SE Category	SE Value	Magnitude	Total SE Value	
Field	0.41	238	97.58	
Total Effort			97.58	person-hours

Formulating the 4GT Model Equation

With the calculation of the expansion factor we can now formulate the actual equation of the 4GT Model for estimation projects at Great-West Life.

$$E = 3.10 * [(SF_{form}*0.13)+(BSE_{form}*0.29)+(DSE_{form}*1.59)+(UE_{form}*22.3)+ (SF_{report}*0.13)+(BSE_{report}*0.84)+(DE_{report}*2.55)+(DE_{data}*0.41)]$$

Note that the effort due to process functions is excluded in the basic equation. If some 3GL coding is to occur, the equation is extended by adding the process component (which is estimated directly by the project manager or the developer. The new equation would then appear as:

$$E = 3.10 * [(SF_{form}*0.13)+(BSE_{form}*0.29)+(DSE_{form}*1.59)+(UE_{form}*22.3)+ (SF_{report}*0.13)+(BSE_{report}*0.84)+(DE_{report}*2.55)+(DE_{data}*0.41)+(process)]$$

Note that E can be refined further using PFES if so desired. PFES evaluates project factors such as "skill of programmer", "environment" and "application characteristics".

Conclusion: The above formula is modular and therefore can provide more accurate estimates as a project progresses. This is a desirable quality in a cost model as very little information is generally available at the start of a project. Connell and Shafer describe the advantages of a modular formula:[60]

> We would like the formulas to be modular so actual numbers derived from measured performance can be plugged in at the end of each project phase, thus steadily improving the accuracy of the estimates as the project moves toward completion.

As indicated earlier on the 4GT Model uses software metrics (past measurements) to assist with the determination of the expansion factor. The systems dynamics model researched by Abdel-Hamid[22,23] (see Chapter 3) indicates that one should be careful about using such historical data for calibrating new models — he indicates that we can show that:

> a software estimation tool cannot be adequately judged only on how accurately it matches historical project results and ... a more accurate estimate is not necessarily a "better" estimate.

He explains that "a different estimate creates a different project" and that a model should therefore be judged not only on the basis of how accurate it is but also if the its estimates are "not costly".

In our experimentation we took the above facts into consideration. The data used by for experimentation was checked for normality. Also, additional steps were taken to ensure that only reliable metrics are used — according to Pressman, if a metrics baseline consisting of data collected from past software development projects can be established, several benefits can be obtained for cost and effort estimation modelling purposes but the data must have the following attributes: data must be reasonably accurate; measurements must be consistent; applications must be similar to work that is to be estimated (page 58).[56] In developing the expansion factor for Great West Life the above guidelines were complied with fully.

5.5 Validation of the Effort Equation

With the determination of the expansion factor for Great-West Life (using LEGASY) we next proceeded with using the 4GT Model for testing other projects. The *Telephone System* project developed for the Communications Dept at Great West Life (using ORACLE) was used as a detailed test case. The project used the same life cycle as Legasy. The participants of this project were Minaker and Davis. Several meetings occured with Minaker who was the project manager for this project (see Table 3.1). The Telephone Project was suitable for validation purposes for several reasons:

- The project was small and relatively straight forward.
- The essential input data, required for estimation purposes, was readily available.

- It also did not have a 3GL programming component. Therefore effort due to Process Functions did not have to be estimated.

The computations relevant to this project are presented below. It reveals that the estimated effort was close to the actual. The effort estimated using the 4GT Model equation turned out to be 171 person hours, and the actual implementation effort for the system was 160 person hours.

Effort Estimation for the Telephone System using the 4GT Model				
Form			27.89	person-hours
Report			7.05	person-hours
Data			20.09	person-hours
Process			0.00	person-hours
Estimated Development Effort [D]			55.03	person-hours
Expansion Factor (for GWL)			3.1	
Estimated Effort [E]			171	person-hours
Project Factors Correction [PF]			1.00	
Adjusted Estimated Effort [E adj.]			171	person-hours
Actual Effort Determined on Project Completion			160	person-hours
FORM				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	101	13.13	
Basic SE	0.29	18	5.22	
Detailed SE	1.59	6	9.54	
User Exit	22.73	0	0.00	
Total Effort			27.89	person-hours

REPORT				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	15	1.95	
Basic SE	0.84	0	0.00	
Detailed SE	2.55	2	5.10	
Total Hours			7.05	person-hours
DATA				
SE Category	SE Value	Magnitude	Total SE Value	
Data Element	0.41	49	20.09	
Total Hours			20.09	person-hours

5.6 A Case Study

A case study was conducted at the University of Winnipeg to study portability issues related to the model. Hildebrand, Systems Coordinator, Pitblado & Hoskin[82] provided assistance with training the subjects, etc. The following were our objectives here:

- validate the model weights further.
- investigate if SEs can be unambiguously classified into the various categories by estimators.
- investigate model flexibility in adapting to other environments.

A formal research procedure used by Teng and Jamison[95] to investigate fourth generation query languages was used as a model to conduct the research.

5.6.1 Method Used

A video rental case study was released to twenty-six participants at the University of Winnipeg during the Fall term of 1991. The subjects were senior students and all had average experience with software development using 3GLs and experience with various microcomputer based packages such as dBASE and Lotus 1-2-3. Subjects also had some knowledge of ORACLE (including SQL*PLUS and SQL*FORMS) from a data base management course. However, to ensure that they had full exposure to all relevant ORACLE tools (especially those pertaining to our case study) they were given a two day hands-on session.[82] The objective was to ensure that our subjects can be regarded as good samples for validation purposes. Regardless, all our subjects still only had one to four person-months worth of 4GT tools experience. We therefore felt compelled to rate our subjects as LOW in 4GT tool experience.

The case study involved implementation of a Video Rental System (VRS). Predefined entities such as customer, video, transaction and functions such as add-new-customer, add-new-video, query-customer-form, query-video-form, and rental-transaction were used (See Appendix H describing the various functions and SEs involved). In other words the user requirements and systems analysis and design associated with VRS were done by us. However, everyone was provided with an adequate background of the case study with regards to purpose and scope, as well as inputs, outputs, and processes involved with the system.

The weights described in the 4GT equation of Table 5.1 were used to predict the effort required to implement the various functions of the above case study. The subjects were not given any information or clue as to how much effort it should take to develop the various functions of the systems, as this could have distorted the results. According to research carried out,[23,96] subjects can unwittingly change their normal behaviour to conform with goals set by a researcher.

As it was not our intention to estimate or validate system development effort (E) but only determine development effort (D), we targeted a few functions of the VRS system for estimation purposes. Participants were asked to record the actual time associated with developing data, form and report functions by classifying screen fields into SE categories. A sample questionnaire used at the end of the case study and completed by the subjects is illustrated in Table 5.2.

5.6.2 Results

The screen fields and their SE categories, and the format of the questionnaire used are presented next followed by our conclusions.

Screen Fields and Specification Elements by Categories

Customer function	n= 27
Field Name	SE Category
ID	Basic
Name	Basic
Address	Basic
Credit-Rating	Detailed
Video function	Simple
Vid-Id	Basic
Type	Detailed
Mov-name	Basic
Rental	Basic
Transaction Function	n = 27
ID	Detailed
Name	Basic
Address	Basic
Date	Detailed
ID	Detailed
Vid-id	Basic
Mov-name	Basic
Rating	Basic
Price	Basic
Transaction Report	n = 27
ID	Basic
Name	Detailed
Address	Basic
Date	Basic
ID	Detailed
Vid-id	Detailed
Mov-name	Basic
Rating	Basic

For each of the following questions document accurately the effort in person hours involved with development. Do not include the time spent waiting for a computer, or computer down-time.

A. Data Function

How much time in total did it take to create the three tables? ___

B. Form & Report Function

Circle the category each screen field falls into (S = Simple, B = Basic, D = Detailed) and indicate the time taken to implement the screen field.

Function Name = ______________

Average *Simple* Screen Field = ___

Field#	Name	Category	Effort
1.		S B D	___
2.		S B D	___
3.		S B D	___
4.		S B D	___
5.		S B D	___
6.		S B D	___

Function Name = ______________

Average *Simple* Screen Field = ___

Field#	Name	Category	Effort
1.		S B D	___
2.		S B D	___
3.		S B D	___
4.		S B D	___
5.		S B D	___
6.		S B D	___
7.		S B D	___

Box 5.1 Case Study Questionnaire Format

The average model weights determined by our subjects are illustrated in Table 5.2 below. As expected, the various weights are higher than those determined at the Great-West Life (see Table 5.1) since our subjects were uniformly rated as LOW. To facilitate comparison with the calibrated weights of Great-West Life (where the calibrators had average experience) we had to adjust the model weights downwards. A rating of 1.10 for the *coding* stage was used as a guide to effect this correction (this was also consistent with Boehm (p. 442).[16] Therefore a project factor correction of 0.91 (1/1.10) was applied uniformly to the average weights described in Table 5.2. These results are presented in Table 5.3. It reveals that the model weights are quite close with those determined at Great-West Life.

Table 5.2: Case Study Model Weights Before Correction

Project Participant	Function Type	Simple [SF]	Basic [BSE]	Detailed [DSE]	User-Exit [UE]	Data Element [DE]
UW	Form	0.15	0.31	1.70	N/A	N/A
UW	Report	0.15	0.90	2.75	N/A	N/A
UW	Data	N/A	N/A	N/A	N/A	0.50

Table 5.3: Case Study Model Weights After Correction

Project Participant	Function Type	Simple [SF]	Basic [BSE]	Detailed [DSE]	User-Exit [UE]	Data Element [DE]
UW	Form	0.14	0.28	1.55	N/A	N/A
UW	Report	0.14	0.82	2.50	N/A	N/A
UW	Data	N/A	N/A	N/A	N/A	0.46

Estimated development effort for this case study using the 4GT Model is presented below. It was determined to be 22.84 person hours.

Effort Estimation for the V.R.S Case Study using the 4GT Model				
Form			13.77	person-hours
Report			4.56	person-hours
Data			4.51	person-hours
Process			0.00	person-hours
Estimated Development Effort (D)			22.84	person-hours
F O R M				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	18	2.34	
Basic SE	0.29	12	3.48	
Detailed SE	1.59	5	7.95	
User Exit	22.73	0	0.00	
Total Effort			13.77	person-hours

R E P O R T				
SE Category	SE Value	Magnitude	Total SE Value	
Simple SE	0.13	9	1.17	
Basic SE	0.84	1	0.84	
Detailed SE	2.55	1	2.55	
Total Hours			4.56	person-hours
D A T A				
SE Category	SE Value	Magnitude	Total SE Value	
Data Element	0.41	11	4.51	
Total Hours			4.51	person-hours

With regards to other results, we found that with the exception of two subjects, all subjects classified screen fields into the SE categories correctly — indicating that, at least for the VRS case study, SE classification was straight forward.

5.7 Using the Model in the Early Stages of Feasibility

As indicated in Chapter 3, the model is capable of estimating early on in the life cycle, i.e., during the Feasibility Study & Requirements Definition Phase (see Table 3.2). During this stage the project manager requires very general estimates of the project on hand. Basically, such information will indicate whether the upper management is interested in funding the development of a new software system. This is also called *ball-park* estimating.[106] The 4GT Model can be used at this stage to provide a reasonable effort estimate. The strategy used for determining such an estimate involves estimating the total number of functions of a system.

The 4GT equation for ball-park estimating is:

E = 3.1*[(10.2 * no. of forms)+(7.9 * no. of reports)+(4.9 * no. of data tables)]

The weights 10.2, 7.9 and 4.9 were determined by adding the total development efforts (D) of the LEGASY System and Telephone System, and then dividing by the total number of functions (in corresponding categories). This process can be explained mathematically as follows:

E = 3.1 * [(SF_{form}*0.13)+(BSE_{form}*0.29)+(DSE_{form}*1.59)+(UE_{form}*22.7)] / total no. of form functions + [(SF_{report}*0.134)+(BSE_{report}*0.84)+(DE_{report}*2.55)] / total no. of report functions + [(DE_{data}*0.41)] / total no. of data functions

5.7.1 Testing the 4GT Ball-Park Estimating Equation

We decided to test the above 4GT Ball-Park estimating equation using statistics from the Telephone System.

Total Number of Form Functions = 5

Total Number of Report Functions = 2

Total Number of Data Tables = 3

E = 3.1 * [(10.2 * 5) + (7.9 * 2) + (4.9 * 3)]

= 252 person hours.

On analysis it reveals that this number is higher than the actual effort consumed in the project, which is 160 person-hours. This was not unexpected as the Telephone system does not have any user exits — the user exit SEs constitute a substantial proportion of the effort involved in the Ball-Park equation. On excluding the impact of the user exit SEs - we get a new 4GT ball-park equation.

E = 3.1*[(3.1 * no. of forms)+(7.9 * no. of reports)+(4.9 * no. of data tables)]

We can now use this equation to test the Telephone system project objectively.

E = 3.1 * [(3.1 * 5) + (7.9 * 2) + (4.9 * 3)]

= 142 person-hours.

This estimate can be considered to be a satisfactory ball-park estimate.

5.7.2 Ball-Park Estimating for LEGASY

As the LEGASY system has user exit SEs and process functions we use the following equation to obtain a ball-park estimate of the LEGASY system project.

E = 3.1 * [(10.2 * 5) + (7.9 * 2) + (4.9 * 3) + Process Effort]

Given the following statistics about the LEGASY project:

Total Number of Form Functions = 28

Total Number of Report Functions = 24

Total Number of Data Tables = 17

Total Process function effort = 125 person-hours

E = 3.1 * [285.6 + 189.6 + 83.3 + 125]

= 2119 person-hours

This ball-park estimate compares reasonably well with the actual effort of 2340 person hours for LEGASY. Similarly, for the VRS project, this estimate is 18 person hours vs. the actual 22 person-hours.

5.8 Estimating Effort Under Different 4GT Paradigms

In Chapter 3 we described different 4GT paths (see section 3.3 — systems development methodology). In this section we suggest how the 4GT Model can be used to estimate the development effort under different 4GT paradigms.

5.8.1 Estimating Evolutionary Prototyping Projects

Evolutionary prototyping is popular today.[60] A major reason for the popularity is the fact that good quality code is generated during prototyping. The logic that "a prototype must be discarded as it was developed in a hurry" is generally not valid with 4GT based development, as a substantial amount of the code was automatically generated, and not physically coded (where the chances of human error are high).

As indicated earlier on a lot of experimentation described in this chapter pertains to the "evolutionary prototyping" paradigm. With such an approach, code generation takes place throughout the life cycle, and developed code is not discarded. Great-West Life, for instance, follows the evolutionary prototyping paradigm when using 4GTs. Here **skeleton screens** are developed early on in the life cycle and demonstrated to users who test the human-computer interface. As the project progresses this prototype evolves into the final product. The 4GT Model weights calibrated in this chapter are based on this methodology.

5.8.2 Estimating Throw-Away Prototyping Projects

When throw-away prototyping is involved we can consider two extreme development modes (see Table 2.1):

- "screen" or "simple mock up" prototyping
- "detailed" or "full" prototyping

With *screen* or *simple mock up* prototyping only simple SEs are involved (as alluded to in section 3.5.3.1). Since such a prototype is going to be discarded eventually (and hence the simple SE effort, i.e., SF's in our 4GT equation), we can regard the effort estimation equation for such projects to be:

$$\mathbf{E_{screen} = E + (0.13 * SF)}$$

With *detailed* or *full* prototyping, all the functions are fully developed (demonstrated to the user) and then discarded. The effort equation for estimating such projects should then be:

$$\mathbf{E_{detailed} = E + D}$$

To illustrate with the LEGASY system, if screen prototyping and detailed prototyping were to occur, project estimates using the above equations would be:

$$E_{screen} = E + (0.15 * SF)$$

$$= 2340 + 24.05 = 2364.82 \text{ person-hours}$$

$$E_{detailed} = E + D$$

$$= 2340 + 755.41 = 3095.41 \text{ person-hours}$$

5.8.3 Estimating Non Prototyping Projects

If we can assume that the total development effort D would be the same - regardless of whether it was implemented in one chunk (as in a non-prototyping paradigm) or in several chunks (as in the evolutionary prototyping paradigm), then we can use the following equations to determine such effort:

$$\mathbf{D_{evolutionary\text{-}prototyping} = D_1 + D_2 + D_3 + D_n = D_{non\text{-}prototyping} = D}$$

$$\mathbf{E_{non\text{-}prototyping} = Expansion\ factor_{non\text{-}prototyping} * D,\ and}$$

$$\mathbf{E_{evolutionary\text{-}prototyping} = Expansion\ factor_{evolutionary\text{-}prototyping} * D.}$$

In conclusion, we note that the 4GT model is quite versatile. For example, it is capable of supporting the various 4GT paradigms used in the industry today. As data pertaining to all 4GT paradigms is not available readiy, intensive validation of the concepts presented here is left for future research.

5.9 Evaluating Model Portability

According to DeMarco[1] there are no transportable cost models, i.e., a model calibrated at one site cannot be used at another site without modifications. In the case of the 4GT Model, the function development weights (associated with "D") are quite portable for a given 4GT, however, the *expansion ratio* might need to be re-calibrated at another site as it deals with attributes such as project management, communication, etc., which vary from one shop to another. Using a "local" expansion ratio is therefore quite desirable if the 4GT Model is to be used at another site. The procedure described in this chapter can be used to obtain a new expansion ratio.

The various project factors associated with PFES also serve to make our cost model portable. The list of project factors identified during our knowledge engineering process with PFES, however, is not exhaustive. Additional factors, especially organizational, such as "staffing and manpower-acquisition" variables[68] can be incorporated to make the model more portable.

In this context, Abdel-Hamid and Madnick indicate that "the portability of software estimation models can be significantly improved by taking into consideration not only technical aspects of the software development environment", but also, "managerial and organizational characteristics of the environment".[68] However, our research effort has not focussed on these issues yet, the above issues are more relevant when staffing, scheduling, etc. come into picture — these we leave for future research (in association with PFES).

5.10 Conclusions

In this Chapter we described the calibration and experimentation results pertaining to the 4GT Model. The results indicate that our development weights are quite reliable and portable. We also demonstrated how ball-park estimation and base-line estimating are facilitated using the 4GT Model. Finally, we described equations that are useful for cost estimating the various 4GT paradigms.

Chapter 6

Conclusions and Future Directions

6.1 Introduction

This chapter provides conclusions related to modelling fourth generation effort. Future directions related to research in this area are also described.

6.2 Summary of the Results

Here we review the thesis by answering the following questions: What is the problem? How was it tackled? What results were obtained? What is new and better about it?

Our primary objective was to deal with the problem of estimating software development effort when *fourth generation tools* are used. We indicated that traditional predictors or cost models are inadequate for measuring development effort involving 4GTs. Such traditional predictors or models are more oriented towards "physical coding" rather than "specification-oriented coding".

To tackle this problem we introduced a new predictor called *specification element* (SE). SE was defined as a specification task associated with implementing a screen field or data element. SEs were categorized logically into a few distinct and manageable categories based on the nature of specification effort involved.

SEs operate on functions — the following functions were identified for the fourth generation environment: form, report, data, and process. As functions and screen fields can be counted easily (even early on in the life cycle), SEs can be regarded as good predictors.

Each SE has an effort value in person-hours associated with it. This represents the work effort required to implement one SE and hence one screen field. By using the techniques described in the model one can directly determine the overall development effort and adjust it, if necessary, for the influence of *project factors* (such as familiarity with tools, and programmer experience).

Two approaches were taken to ensure that the model was satisfactory. First, the model was evaluated by practitioners and then installed in a large commercial setting. It was calibrated using actual project data and tested against a new project. The results obtained here revealed that the model resolved various effort estimation problems satisfactorily.

The second approach taken was to determine if the weights obtained above are suitable for use at another site. An experimental project (case study) implemented by several subjects was used to test the model. Results obtained here revealed that the model weights, especially those related to development effort (D) were quite portable.

We have realized the following objectives with the 4GT Model:

1. The model provides us a basis to measure specification oriented application development effort. Application generators, form generators and report generators can all now be measured satisfactorily.

2. The model supports sizing of 4GT applications in two modes — *ball-park* and *base-line*. While ball-park sizing provides us rough estimates early on in the system development life cycle, base-line sizing provides us detailed estimates on completion of some software design. Base-line therefore provides us better estimates than ball-park sizing.

3. The model supports various 4GT development paradigms for purposes of effort estimation. The 4GT Model itself is based on the evolutionary paradigm, as such, its ability to measure projects based on this paradigm has been extensively tested. The *expansion factor* used by the

model is capable of supporting the other special cases of 4GT development.

4. Finally, we experimented with the knowledge-based method for adjusting project effort. We found some interesting advantages with such an approach:
   - It was easy to perform what-if analysis — thanks to the built-in user friendly interface of the expert system shell. Various scenarios were experimented with for purposes of estimation - "allowing end-users to compute vs. letting data processing staff compute"; "using resources with average skill vs. using resources with very high skill"; and "putting skilled people on a difficult project vs. putting skilled people on an easy project".
   - New rules and project factors governing end-user computing were researched and introduced in the model.

6.3 Future Work

This topic is examined under the following two headings — model enhancement, and model integration.

6.3.1 Model Enhancement

The following three points need to be researched in the near future:

1. *Calibration of the process function:* As evident in the previous chapter, research was not conducted to calibrate the process function of the 4GT Model for ORACLE. (Only the form, report, and data functions were calibrated.) Even though it is not necessary to calibrate the process function (as one might continue to see 4GT projects such as the Telephone System project that do not need process functions), it is still useful to investigate this problem. Any experimentation here should clearly focus on non-procedural process functions only as no model for estimating such effort exists. Wrigley and Dexter appear to be working towards implementation of one such a model but their work is yet incomplete — they have established links between information system size and non-procedural LOCs but they have not associated such LOCs with effort.

2. *Calibration of other popular fourth generation tools:* Calibrating ORACLE was worthwhile — as evidenced by the interest shown by practitioners in using the 4GT Model and the fact that it is a very popular relational system, listing second only to DB2, in popularity (p.13)[51] — nevertheless, it will be worthwhile to calibrate other 4GTs as well. This would enable us to compare the 4GT weights calibrated for ORACLE with those in other environments.

3. *Using the model to test the different 4GT paradigms*: Our research and the various equations presented in the context of throw-away prototyping vs. evolutionary have raised some questions. Is the expansion factor the same for both of these paradigms? What about the Spiral Model? Which one is a more expensive paradigm? Throw-away, or evolutionary (i.e., which expansion factor is more costly?). The 4GT Model can probably be used as an experimental vehicle to answer such questions.

The above research can be facilitated if a formal software metrics program is established at several sites involved with fourth generation development. Very few sites at present have a software metrics program. From our perspective, however, it is important to gather the following data from several sites:

- type of 4GT paradigm used
- project start and completion dates
- manpower allocation for the project (including nature of involvement — such as part-time or full time, etc.)
- total number of functions developed by type (i.e., form, report, etc.)
- total number of screen fields involved with each function by SE type.
- effort used to develop each function
- initial effort estimates & the actual project effort

In such a software metrics database, it is preferable to store normalized data only — especially, data pertaining to *actual project effort* should be corrected for overestimation or underestimation. Project overestimation can be costly, according to Abdel-Hamid, "wasteful project practices such as goldplating", and "unproductive slack time activities" can occur with such projects — underestimation of projects can also be costly as it results in an "initial understaffing, followed by a costly staff buildup later in the life cycle".[22] Using such data as a benchmark in future model calibrations (such as for determining the *expansion factor*) can result in costly projects.[22]

In conclusion we note that good metrics data can provide us with substantial benefits - it can assist us with: future project planning, future cost modelling, and resolving model portability issues. In this sense, we agree with Yourdon, who states that, "software metrics can be as valuable a *silver bullet* for your organization as CASE technology, structured techniques, or fourth generation languages" (p. 10).[14]

6.3.2 Model Integration

One of the key advantages of the 4GT Model is that it lends itself readily to integration within the CASE architecture. CASE tools use a central repository for storing project data. Such a repository — commonly called *CASE Database* — could be easily tapped to obtain data pertaining to the development process.

Unfortunately, by design, existing cost models are stand-alone products, i.e., they do not integrate with CASE tools or store or tap into a CASE database for estimation or planning purposes. If such cost models interface, or fully integrate with a CASE database, we can realize several benefits. For instance, with the 4GT Model, the following input sizing parameters required by the model for producing ball-park or base-line estimates can be directly supplied from the database:

- total number of tables
- total number of data elements
- total number of reports and forms
- total number of screen fields

- total number of process functions.

The advantages of integrating CASE tools with cost models are two fold:

1. The estimation process can now be automated to a higher degree.
2. Historic software metrics data can be retrieved easily for calibration purposes. As CASE projects are always up-to-date any data captured would also be very accurate.

At present we are experimenting with an integrated CASE architecture at the Great-West Life — it involves the *4GT Model* and *PFES* functioning as "CASE estimating tools" and a "CASE database".

6.4 Conclusion

In the final note of his book Controlling *Software Projects,*[1] DeMarco exclaims:

> "Good grief, it's the end of the book. Have I made my point? Was the meaning clear? ... Isn't there *much, much* more to say about software quality and function weighting and complexity measurement and organization of the development process and ...?"

Well, I conclude with almost similar anxiety, fourth generation software development and effort estimation are fascinating topics and indeed there is *much, much* more to say here as well — scheduling, software prototyping metrics and models, complexity measurement, risk analysis and management, neural nets for effort correction — and much more come to mind. However, I have been advised to leave these topics aside for a future encounter ... and wisely so.

References

1. DeMarco, T., *Controlling Software Projects*, (foreword by Boehm B.), Yourdon Press Computing Series, Prentice-Hall, Englewood Cliffs, 1982.

2. Martin, J., *Fourth-Generation Languages*, Vol. I, Prentice-Hall, 1985.

3. Bate, J., Vadhia, D., *Fourth Generation Languages Under DOS and UNIX*, BSP Professional Books, 1987.

4. Chorafas, D., *Fourth and Fifth Generation Programming Languages*, McGraw-Hill Inc., Vol I, 1986.

5. Pressman, R. *Software Engineering A Practitioners Approach*, Second Edition, McGraw-Hill Book Company, 2nd Edition, 1987.

6. Pressman, R. *Software Engineering A Practitioners Approach*, Third Edition, McGraw-Hill Book Company, 1992.

7. Martin, J., *Application Development Without Programmers*, Prentice-Hall, Englewood Cliffs, NJ, 1982, p.30.

8. Lin, C.,"Systems Development with Application Generators: An End-User Perspective," *Journal of Systems Management*, Vol 41, No.4, 1990, pp.32-36.

9. Martin, M.,"Instant Screen Design," *Journal of Systems Management*, Vol 41, No.4, 1990, pp.22-27.

10. Verner, J., G. Tate, "Estimating Size and Effort in Fourth-Generation Development," *IEEE SOFTWARE*, July 1988, pp 15-22.

11. Grady, R., Work-Product Analysis: The Philosopher's Stone of Software, *IEEE Software*, March 1990, p. 26-34.

12. Mills, Harlan, P. Dyson, "Using Metrics to Quantify Development," *IEEE Software*, March 1990, p 15-16.

13. Grady, R., D. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.

14. Ed Yourdon, "Software Metrics: You Can't Control What You Can't Measure," *American Programmer*, Vol 2, No. 2, February 1989.

15. Schussel, G., "Fourth Generation Productivity Tools - A Shopping Guide for Software Consumers," *Data Management*, October 1984, pp. 42-46.

16. Boehm, B., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, Prentice-Hall Inc., 1981.

17. Matos, V.M., and Jalics, P.J, "An Experimental Analysis of the Performance of Fourth Generation Tools on PCs," *Communications*, ACM 32, 11, Nov. 1989, 1340-1351.

18. Albrecht, A.J., "Measuring Application Development Productivity," *Proc. IBM Application Development Symposium*, Monterey, CA, Oct. 1979, pp. 83-92.

19. Dreger, B.J., *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

20. Abdel-Hamid, T.K., "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective," Unpublished Ph.D. dissertation, Sloan School of Management, MIT, January, 1984.

21. Abdel-Hamid, T.K., "Investigating the Cost/Schedule Trade-Off in Software Development," *IEEE Software*, pp. 97-105, January 1990.

22. Abdel-Hamid, T.K., "On the Utility of Historical Project Statistics for Cost & Schedule Estimation: Results from a Simulation-Based Case Study," The *Journal of Systems and Software*, 1990.

23. Abdel-Hamid, T.K, Madnick, S.E, *Dynamics of Software Project Management*, Prentice-Hall, Englewood Cliffs, N.J, 1991.

24. Verner, J., Tate, G., "Estimating Size and Effort in 4GL development," *IEEE Software*, July 1988, pp. 15-22.

25. Dreger, B.J, *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, p 132, 1989.

26. Wrigley, C., Dexter. A., "A Model for Measuring Information System Size," *MIS Quarterly*, June 1991, 245-257.

27. Waterman, D., *A Guide to Expert Systems*, Addison Wesley Publishing Co, 1986.

28. Biegel, J., Bearden, M., Dickerson,D., O'Donnell, "Building an Expert System for Cost Estimating," *International Industrial Engineering Conference Proceedings*, 1986, pp. 504-509.

29. Arrowood, L., Emrich, M., Sadlove,R., Jones, A., Watson, B., Suprapaneni, R., "Knowledge-Based vs Traditional Cost Estimation Models," (reprint, US Department of Energy), November 1989, *Datapro Research*, McGraw-Hill Inc., AS20-050, Nov. 1989, pp.201-207.

30. Ntuen, C., Mallik, A., "Applying Artificial Intelligence to Project Cost Estimating," *Cost Engineering*, Vol. 29, No.5, May 1987, pp. 8-12.

31. Avots, I., "The Coming Impact of Artificial Intelligence on Project Management," *Project Management in Progress*, North-Holland, 1986, pp. 307-312.

32. Kanabar, V., Seah, E.,Scuse, D., *Knowledge-base Referencing During Planning*, Working Papers on Artificial Intelligence in Management Science, The Institute of Management Sciences, Fall 1989, pp. 144-56.

33. Verner, J., Tate, G., "Estimating Size and Effort in Fourth-Generation Development," *IEEE Software*, July 1988, p.15-22.

34. Matos, V.M, Jalics, P.J., "An Experimental Analysis of the Performance of Fourth Generation Tools on PCs," *Communications ACM*, 32, 11, Nov. 1989, 1340-1351.

35. Misra S., Jalics, P., "Third Generation versus Fourth-Generation Software Development," *IEEE Software*, July 1988, p.8-14.

36. Jones C., *Programming Productivity*, McGraw-Hill, 1986.

37. Dredger B., *Function Point Analysis*, Prentice-Hall, 1989, p.12.

38. Symons, C., "Function Point Analysis, Difficulties and Improvements," *IEEE Software Transactions on Software Engineering*, SE-14(1), January 1988, pp. 2-10.

39. Wallace R., Stockenberg J.,Charette R., *A Unified Methodology for Developing Systems*, McGraw-Hill Book Company, 1987.

40. Grady, R., Work-Product Analysis: The Philosopher's Stone of Software, *IEEE Software*, March 1990, p. 26-34.

41. Albrecht, A. J., and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Engineering*, November 1983, pp. 639-648.

42. Arthur, L. J., *Measuring Programmer Productivity and Software Quality,* Wiley-Interscience, 1985, p.23.

43. Walston, C., and C. Felix, "A Method for Programming Measurement and Estimation," *IBM Systems Journal,* vol. 16, no. 1, 1977.

44. Jones, C., *Programming Productivity*, McGraw-Hill Inc, 1986.

45. Mohanty, S., "Software Cost Estimation: Present and Future," *Software Practice and Experience*, 1981, pp. 103-121.

46. Londeix, B., *Cost Estimation for Software Development*, Addison-Wesley Publishing Co., 1987, p. 40.

47. Holsapple, C., Whinston, A., *Business Expert Systems*, Irwin Inc, 1987.

48. Hayes-Roth, F., Waterman, D., Lenat, D., *Building Expert Systems*, Addison-Wesley Publishing Co., MA, 1983.

49. Rockart, J.F, Flannery, L., "The Management of End User Computing," *Communications of the ACM*, Association of Computing Machinery, October 1983, pp. 776-784.

50. Putnam, L.H, Fitzsimmons A., "Estimating Software Costs," *Writings of the Revolution*, Yourdon Press, New York, 1982, pp. 326-344.

51. Mackowiak, K., "Skills Required and Jobs Available for CIS Majors", *Interface*, Vol. 13, No.4, 1991, pp. 9-14.

52. Henry, S., Selig, C., "Predicting Source Code Complexity at the Design Stage," *IEEE Software*, March, 1990, pp.36-44.

53. Henry, S.M., Kafura, D., "Software Structure Metrics Based on Information Flow," *IEEE Trans. Software Engg.*, Sept. 1981, pp. 510-518.

54. McCabe, T.J, "A Complexity Measure," *IEEE Trans. Softw. Engg.*, SE-2,4, Dec. 1976, p. 308.

55. SQL*FORMS, "SQL*Forms Class Notes," *ORACLE Corporation*, August 1987.

56. Pressman, R. *Software Engineering A Practitioners Approach*, Second Edition, McGraw-Hill Book Company, 3rd Edition, 1992 (to be published).

57. SQL*FORMS Designer's Reference, Version 2.0, *ORACLE Corporation*, Part No. 3304-V2.0. February 1988.

58. McFadyen, R., Kanabar, V., *An Introduction to Structured Query Language*, Wm. C. Brown, Dubuque, IA, 1991.

59. Wrigley, C., Dexter. A., "A Model for Measuring Information System Size," *MIS Quarterly*, June 1991, pp. 245-257.

60. Connell, J., Shafer, B., *Structured Rapid Prototyping An Evolutionary Approach*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ, 1989.

61. DeMarco, T., *Concise Notes on Software Engineering*, Yourdon Inc., New York, NY, 1979.

62. Fertuck, L., *Systems Analysis and Design with CASE Tools*, Wm. C. Brown, Dubuque, IA, 1992.

63. Avison, D., Fitzgerald, G., *Information Systems Development Methodologies Techniques and Tools*, Blackwell Scientific Publications, 1988.

64. Gore, M., Stubbe, J., *Elements of Systems Analysis*, Fourth Edition, Wm. C. Brown, Dubuque, IA, 1988.

65. Parkin, A., *System Management*, Edward Arnold Publishers Ltd., London, 1980.

66. Clarke, R., "A Contingency Approach to the Application Software Generations", *Data Base*, Summer 1991, pp. 23-34.

67. Itakura, M., Takayanagi, A., "A Model for Estimating Program Size and its Evaluation," *Proceedings of the Sixth International Conference on Software Engineering*, IEEE, 1982, pp. 104-109.

68. Abdel-Hamid, T.K., Madnick, S.E, *Software Project Dynamics An Integrated Approach*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

69. Bailey, J., Basili, V., "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*, IEEE, 1981, pp. 107-116.

70. Kanabar, V., "Knowledge-based Project Management", Department of Mathematics and Statistics Seminar, January 1988.

71. Kanabar, V., "Knowledge-based Project Management: Work-Effort Estimation", *Twentieth Interface Symposium*, Washington, D.C, April 21-23, 1988,

72. Kanabar, V., "An Integrated Software Metrics Model for Planning", Department of Mathematics and Statistics Seminar, January 15, 1989.

73. Kanabar, V., B. Feiring, D. Scuse, E. Seah, "Project Planning and Estimating Using a Software Metrics Framework", *International Conference on Computing and Information*, Canadian Scholars Press, Volume II, 1989, pp. 336-339.

74. Kanabar, V., "An Integrated Model for Automated Planning and Estimation", *Proceedings of ACM Seventeenth Annual Computer Science Conference*, February 1989.

75. October 1989, Faculty Seminar Series, University of Manitoba, Faculty of Management, Effort Estimation of Fourth Generation Languages.

76. Kanabar, V., Seah, E., "A Model for Planning and Cost Estimation", *Advances in Computing and Information*, Proceedings, Niagara, Ontario, 1990, pp. 168-70.

77. Kaehler, N., Software Development, Information Systems and Data Processing Department, Investors Group, interviews and testing conducted between April 1990 and July 1991.

78. Smith, W., Computer Systems, Great-West Life, interviewed on 6/6/91, 6/7/91, 6/12/91, 6/18/91, 6/25/91, 7/2/91, 7/9/91, 7/11/91, 7/12/91, 8/9/91, 15/9/91, 22/9. Subsequently about seven informal consultations until 12/6/91.

79. Garner, T., Computer Systems, Great-West Life, interviewed on 5/13/91, 5/20/91, 6/6/91, 6/7/91, 6/12/91, 7/11/91, 8/9/91, 22/9. Subsequently about three informal consultations until 12/6/91.

80. Buskens, R., Computer Systems, Great-West Life, interviewed on 6/10, 6/18, 6/25.

81. Hildebrand, M., Pitblado & Hoskin, model calibration on 7/11/91, 7/13/91, 7/15/91. Informal consultations until 12/6/91.

82. Hildebrand, M., Pitblado & Hoskin, Winnipeg, 4GT calibration & case study participation on 8/28/91, 8/29/91 & 9/3/91.

83. Allison, M., (Telephone System project manager), Computer Systems, Great-West Life, calibration meeting on 9/9/91.

84. Layer., A., Quality Assurance, Information Systems and Data Processing Department, Investors Group, data collection between April 1990 and December 1990.

85. DeMarco, T., & Lister, L. (1990). *Software State-of-the-Art: Selected Papers.* New York, NY: Dorset House Publishing.

86. Kanabar, V., Janzen, T.,Seah, E., Smith, W., "Installation of a 4GT Model", *Technical Report*, Faculty of Management, 1991.

87. Kanabar, V., "CASE: Integrating Project Estimating Tools into the Architecture", Chapter published in *CASE Issues for the 1990's*, ed. Bergin, T., Idea Book Publishing, 1992.

88. Yuen, K. (Project Leader), Chan, E., Ho, C., Ng, K.H., "Space Accounting System", University of Winnipeg, Winnipeg, 1989.

89. Lau, K., (Project Leader), Chan, T., Lam, A., Lam, C. "Trackers", University of Winnipeg, Winnipeg, 1989.

90. Fox, G. (Project Leader), Lee, P., Siu, S., Yap, G., "Mayday Project", University of Winnipeg, Winnipeg, 1989.

91. Finlay, I. (Project Leader), Brandt, G., Tang, A., Zirdum, A., "Weights and Measures Microcomputer System", University of Winnipeg, 1988.

92. Kanabar, V., "Estimating Software Development using Fourth Generation Tools", Naval Postgraduate School, Monterey, California, October, 1990.

93. Kanabar, V., "A Model for Estimating Software Development Effort using Fourth Generation Tools", University of South Florida, Tampa, Florida, Department of Computer Science and Engineering Seminar, April, 1991.

94. Pressman, R., *Making Software Engineering Happen*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

95. Teng, J., Wesley, J., "User Evaluation of Database Query Languages: A Comparison of SQL and DBASE III," *INFOR*, Vol. 28, August 1990.

96. Armitage, H., *The Choice of Productivity Measures in an Organization*, The Society of Management Accountants of Canada, 1991.

97. Roetzheim, W., *Structured Computer Project Management*, Prentice Hall, Englewood Cliffs, New Jersey, 1988, pp. 92-95.

98. Damodaran, M., "Fourth Generation Tools - Characteristics, Applications and their Evolution", *First International Workshop on Computer-Aided Software Engineering*, Volume I, pp. 157-159.

99. Peters, T., Waterman, R. *In Search of Excellence*, N.Y: Harper & Row, 1982, p. 240.

100. Uniface, *Uniface V. 5.2.* Uniface Corporation. Alameda:CA, 1989.

101. Ingres, *Ingres/Applications*. Relational Technology Inc. Alameda: CA, 1986.

102. dBASE IV, *dBASE IV Documentation*, Aston Tate Corporation, CA, 1990.

103. Silver, A., "On the Structural Decomposition and Hierarchical Recombination of Non-Directed Linear Graphs using Multi-Attribute Agglomerative Polythetic Clustering Metrics," *Constructive Approaches to Mathematical Models Symposium*, Carnegie-Mellon University, July 10-14, 1978.

104. Silver, A., "Structural Decomposition using Entropy Metrics," *Proceedings of the 1978 conference on Information Sciences and Systems*, John Hopkins University, March 1978.

105. Corner, R., *Business Systems Design and Development*. Englewood-Cliffs, NJ: Prentice-Hall, 1990.

106. Putnam, L., Myers, W., Measures for Excellence: Reliable Software on Time, Within Budget, Englewoods Cliffs, New Jersey, 1992.

107. Case, A. Jr. (1986). *Information Systems Development: Principles of Computer-Aided Software Engineering*. Englewood Cliffs, N.J: Prentice-Hall.

108. Kolida, G., Assistant Manager, Investors Group, interviews between May 1989- August 1990.

109. IBM, *Managing Projects with Application System*, Release 4, Product No. 5767-001, 1986.

110. Kemerer, C, "Software Cost Estimation Models", Forthcoming in Software Engineers Reference Book, Surrey, U.K: Butterworth.

111. McClure, C. *CASE is Software Automation*. Englewood-Cliffs, N.J: Prentice-Hall, 1989.

112. Microsoft. *Microsoft Project document: Project Scheduling and Reporting Program*, No.410720011-400-R00-0887, Part No. 00163, 1987.

113. Nastec, *Nastec CASE 2000*, Nastec Corporation, Southfield, Michigan, 1986.

114. Pfleeger, S., *Software Engineering: The Production of Quality Software,* Second Edition, N.Y: Macmillan, 1991.

115. Symantec, *Time Line: The Corporate Choice for Project Management and Presentations*, User Manual, Part # 03-30-00016, 1990.

116. Whitten, J., & Bentley, L., *Using Excelerator for Systems Analysis and Design*, Boston, Irwin, 1987.

117. Crossman, T., "Taking the Measure of Programmer Productivity", *Datamation*, 1979, pp. 144-147.

118. Hicks, J., *Information Systems in Business: An Introduction*, Second Edition, West Publishing Co., St.Paul, 1990.

119. Vicinanza, S., Mukhopadhyay, T., Prietula, M., "Software-Effort Estimation: An Exploratory Study of Expert Performance," *Information Systems Research*, December, 1991, pp. 243-262.

120. Ramsey, C., Basili, V., "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, 15, 1989, pp. 747-759.

Bibliography

Abdel-Hamid, T.K., Madnick, S.E, *Software Project Dynamics An Integrated Approach*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Abdel-Hamid, T.K., "On the Utility of Historical Project Statistics for Cost & Schedule Estimation: Results from a Simulation-Based Case Study," *The Journal of Systems and Software*, 1990.

Abdel-Hamid, T.K., "Investigating the Cost/Schedule Trade-Off in Software Development," *IEEE Software*, pp. 97-105, January 1990.

Abdel-Hamid, T.K, Madnick, S.E, *Dynamics of Software Project Management*, Prentice-Hall, Englewood Cliffs, N.J, 1991.

Abdel-Hamid, T.K., "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective," Unpublished Ph.D. dissertation, Sloan School of Management, MIT, January, 1984.

Albrecht, A.J., "Measuring Application Development Productivity," , *Proc. IBM Application Development Symposium*, Monterey, CA, October 1979, pp. 83-92.

Albrecht, A. J., and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Engineering*, November 1983, pp. 639-648.

Armitage, H., *The Choice of Productivity Measures in an Organization*, The Society of Management Accountants of Canada, 1991.

Arthur, L. J., *Measuring Programmer Productivity and Software Quality,* Wiley-Interscience, 1985, p.23.

Arrowood, L., Emrich, M., Sadlove,R., Jones, A., Watson, B., Suprapaneni, R., "Knowledge-Based vs Traditional Cost Estimation Models," (reprint, US Department of Energy), November 1989, *Datapro Research*, McGraw-Hill Inc., AS20-050, Nov. 1989, pp.201-207.

Avison, D., Fitzgerald, G., *Information Systems Development Methodologies Techniques and Tools*, Blackwell Scientific Publications, 1988.

Avots, I., "The Coming Impact of Artificial Intelligence on Project Management," *Project Management in Progress*, North-Holland, 1986, pp. 307-312.

Bate, J., Vadhia, D., *Fourth Generation Languages Under DOS and UNIX*, BSP Professional Books, 1987.

Bailey, J., Basili, V., "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*, IEEE, 1981, pp. 107-116.

Biegel, J., Bearden, M., Dickerson,D., O'Donnell, "Building an Expert System for Cost Estimating," *International Industrial Engineering Conference Proceedings*, 1986, pp. 504-509.

Boehm, B., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, Prentice-Hall Inc., 1981.

Case, A. Jr. (1986). *Information Systems Development: Principles of Computer-Aided Software Engineering*. Englewood Cliffs, N.J: Prentice-Hall.

Chorafas, D., *Fourth and Fifth Generation Programming Languages*, McGraw-Hill Inc., Vol I, 1986.

Clarke, R., "A Contingency Approach to the Application Software Generations", *Data Base*, Summer 1991, pp. 23-34.

Connell, J., Shafer, B., *Structured Rapid Prototyping An Evolutionary Approach*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ, 1989.

Corner, R., *Business Systems Design and Development*. Englewood-Cliffs, NJ: Prentice-Hall, 1990.

Crossman, T., "Taking the Measure of Programmer Productivity", *Datamation*, 1979, pp. 144-147.

Damodaran, M., "Fourth Generation Tools - Characteristics, Applications and their Evolution", *First International Workshop on Computer-Aided Software Engineering*, Volume I, pp. 157-159.

dBASE IV, *dBASE IV Documentation*, Aston Tate Corporation, CA, 1990.

DeMarco, T., Lister, L. (1990). *Software State-of-the-Art: Selected Papers.* New York, NY: Dorset House Publishing.

DeMarco, T., *Concise Notes on Software Engineering*, Yourdon Inc., New York, NY, 1979.

DeMarco, T., *Controlling Software Projects*, Yourdon Press Computing Series, Prentice-Hall, Englewood Cliffs, 1982.

Dreger, B.J., *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

Fertuck, L., *Systems Analysis and Design with CASE Tools*, Wm. C. Brown, Dubuque, IA, 1992.

Gore, M., Stubbe, J., *Elements of Systems Analysis*, Fourth Edition, Wm. C. Brown, Dubuque, IA, 1988.

Grady, R., Work-Product Analysis: The Philosopher's Stone of Software, *IEEE Software*, March 1990, p. 26-34.

Grady, R., D. Caswell, *Software Metrics: Establishing a Company-Wide Program,* Prentice-Hall, 1987.

Hayes-Roth, F., Waterman, D., Lenat, D., *Building Expert Systems*, Addison-Wesley Publishing Co., MA, 1983.

Henry, S.M., Kafura, D., "Software Structure Metrics Based on Information Flow," *IEEE Trans. Software Engg.*, Sept. 1981, pp. 510-518.

Hicks, J., *Information Systems in Business: An Introduction*, Second Edition, West Publishing Co., St.Paul, 1990.

Holsapple, C., Whinston, A., *Business Expert Systems*, Irwin Inc, 1987.

IBM, *Managing Projects with Application System*, Release 4, Product No. 5767-001, 1986.

Ingres, *Ingres/Applications*. Relational Technology Inc. Alameda: CA, 1986.

Itakura, M., Takayanagi, A., "A Model for Estimating Program Size and its Evaluation," *Proceedings of the Sixth International Conference on Software Engineering*, IEEE, 1982, pp. 104-109.

Jones C., *Programming Productivity*, McGraw-Hill, 1986.

Kanabar, V., Seah, E., "A Model for Planning and Cost Estimation", *Advances in Computing and Information*, ICCI Proceedings, Niagara, Ont, 1990, pp. 168-70.

Kanabar, V., B. Feiring, D. Scuse, E. Seah, "Project Planning and Estimating Using a Software Metrics Framework", *International Conference on Computing and Information*, Volume II, Canadian Scholars Press, 1989, pp. 336-339.

Kanabar, V., Janzen, T.,Seah, E., Smith, W., "Installation of a 4GT Model", *Technical Report*, Faculty of Management, 1991.

Kanabar, V., "Knowledge-based Project Management: Work-Effort Estimation", *Twentieth Interface Symposium*, Washington, D.C, April, 1988.

Kanabar, V., "An Integrated Model for Automated Planning and Estimation", *Proceedings of ACM Seventeenth Annual Computer Science Conference*, February 1989.

Kanabar, V., "CASE: Integrating Project Estimating Tools into the Architecture", Chapter published in *CASE Issues for the 1990's*, ed. Bergin, T., Idea Book Publishing, 1992.

Kanabar, V., Seah, E.,Scuse, D., *Knowledge-base Referencing During Planning*, Working Papers on Artificial Intelligence in Management Science, The Institute of Management Sciences, Fall 1989, pp. 144-56.

Kemerer, C, "Software Cost Estimation Models", Forthcoming in *Software Engineers Reference Book*, Surrey, U.K: Butterworth.

Lin, C.,"Systems Development with Application Generators: An End-User Perspective," *Journal of Systems Management*, Vol 41, No.4, 1990, pp.32-36.

Londeix, B., *Cost Estimation for Software Development*, Addison-Wesley Publishing Co., 1987, p. 40.

Mackowiak, K., "Skills Required and Jobs Available for CIS Majors", *Interface*, Vol. 13, No.4, 1991, pp. 9-14.

Matos, V.M, Jalics, P.J., "An Experimental Analysis of the Performance of Fourth Generation Tools on PCs," *Communications ACM*, 32, 11, Nov. 1989, 1340-1351.

Martin, J., *Application Development Without Programmers*, Prentice-Hall, Englewood Cliffs, NJ, 1982, p.30.

Martin, M.,"Instant Screen Design," *Journal of Systems Management*, Vol 41, No.4, 1990, pp.22-27.

Martin, J., *Fourth-Generation Languages*, Vol. I, Prentice-Hall, 1985.

Mills, Harlan, P. Dyson, "Using Metrics to Quantify Development," *IEEE Software*, March 1990, p 15-16.

McCabe, T.J, "A Complexity Measure," *IEEE Transactions on Software Engineering*, SE-2,4, Dec. 1976, p. 308.

McClure, C. *CASE is Software Automation*. Englewood-Cliffs, N.J: Prentice-Hall, 1989.

McFadyen, R., Kanabar, V., *An Introduction to Structured Query Language*, Wm. C. Brown, Dubuque, IA, 1991.

Microsoft. *Microsoft Project document: Project Scheduling and Reporting Program*, No.410720011-400-R00-0887, Part No. 00163, 1987.

Mohanty, S., "Software Cost Estimation: Present and Future," *Software Practice and Experience*, 1981, pp. 103-121.

Misra, S., Jalics, P., "Third Generation versus Fourth-Generation Software Development," *IEEE Software*, July 1988, p.8-14.

Nastec, *Nastec CASE 2000*, Nastec Corporation, Southfield, Michigan, 1986.

Ntuen, C., Mallik, A., "Applying Artificial Intelligence to Project Cost Estimating," *Cost Engineering*, Vol. 29, No.5, May 1987, pp. 8-12.

Parkin, A., *System Management*, Edward Arnold Publishers Ltd., London, 1980.

Pfleeger, S., *Software Engineering: The Production of Quality Software*, Second Edition, N.Y: Macmillan, 1991.

Pressman, R. *Software Engineering A Practitioners Approach*, Third Edition, McGraw-Hill Book Company, 1992.

Pressman, R., *Making Software Engineering Happen*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Putnam, L.H, Fitzsimmons A., "Estimating Software Costs," *Writings of the Revolution*, Yourdon Press, New York, 1982, pp. 326-344.

Putnam, L., Myers, W., *Measures for Excellence: Reliable Software on Time, Within Budget*, Englewoods Cliffs, New Jersey, 1992.

Ramsey, C., Basili, V., "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, 15, 1989, pp. 747-759.

Rockart, J.F, Flannery, L., "The Management of End User Computing," *Communications of the ACM*, Association of Computing Machinery, October 1983, pp. 776-784.

Roetzheim, W., *Structured Computer Project Management*, Prentice Hall, Englewood Cliffs, New Jersey, 1988, pp. 92-95.

Schussel, G., "Fourth Generation Productivity Tools - A Shopping Guide for Software Consumers," *Data Management*, October 1984, pp. 42-46.

Silver, A., "Structural Decomposition using Entropy Metrics," *Proceedings of the 1978 conference on Information Sciences and Systems*, John Hopkins University, March 1978.

Silver, A., "On the Structural Decomposition and Hierarchical Recombination of Non-Directed Linear Graphs using Multi-Attribute Agglomerative Polythetic Clustering Metrics," *Constructive Approaches to Mathematical Models Symposium*, Carnegie-Mellon University, July 10-14, 1978.

SQL*FORMS Designer's Reference, Version 2.0, *ORACLE Corporation*, Part No. 3304-V2.0. February 1988.

SQL*FORMS, *SQL*Forms Class Notes*, ORACLE Corporation, August 1987. Symantec, *Time Line: The Corporate Choice for Project Management and Presentations*, User Manual, Part # 03-30-00016, 1990.

Symons, C., "Function Point Analysis, Difficulties and Improvements," *IEEE Software Transactions on Software Engineering*, SE-14(1), January 1988, pp. 2-10.

Teng, J., Wesley, J., "User Evaluation of Database Query Languages: A Comparison of SQL and DBASE III," *INFOR*, Vol. 28, August 1990.

Uniface, *Uniface V. 5.2*. Uniface Corporation. Alameda:CA, 1989.

Verner, J., Tate, G., "Estimating Size and Effort in Fourth-Generation Development," *IEEE Software*, July 1988, p.15-22.

Vicinanza, S., Mukhopadhyay, T., Prietula, M., "Software-Effort Estimation: An Exploratory Study of Expert Performance," *Information Systems Research*, December, 1991, pp. 243-262.

Wallace R., Stockenberg J.,Charette R., *A Unified Methodology for Developing Systems*, McGraw-Hill Book Company, 1987.

Walston, C., and C. Felix, "A Method for Programming Measurement and Esti mation," *IBM Systems Journal,* vol. 16, no. 1, 1977.

Waterman, D., *A Guide to Expert Systems*, Addison Wesley Publishing Co, 1986.

Whitten, J., & Bentley, L., *Using Excelerator for Systems Analysis and Design*, Boston, Irwin, 1987.

Wrigley, C., Dexter. A., "A Model for Measuring Information System Size," *MIS Quarterly*, June 1991, 245-257.

Yourdon, "Software Metrics: You Can't Control What You Can't Measure," *American Programmer*, Vol 2, No. 2, February 1989.

Appendix A

```
RUNTIME;

ACTIONS

    NOVELTY_IMPACT=SIGNIFICANT
    GT_TOOL_EXP_IMPACT=SIGNIFICANT
    OV_EXP_IMPACT = SIGNIFICANT

    FIND env_value
    FIND env_value2
    FIND RELY
    FIND D_COMM
    FIND CHANGE
    FIND INTERFACE_COMPLX
    FIND OPER_EASE
    FIND METHOD_VALUE

    FIND GT_TOOL_EXP_VALUE
    FIND NOVELTY_VALUE
    FIND TECHNIQUE_EFFECT
    FIND PRACTICE_EFFECT
    FIND IMPACT
    FIND PERSONNEL_PF


    DISPLAY "The 4GT tool value is {GT_TOOL_EXP_VALUE}."
    DISPLAY "The development teams familiarity with the
            application on hand is {NOVELTY_VALUE}."
    DISPLAY "The Personnel PF Correction is {PERSONNEL_PF}."
    DISPLAY "The Methodology Correction is {METHOD_VALUE}."
    DISPLAY "The Application Project Factor Correction is {AF}."
    DISPLAY "The project admin. environment rating is {ENV_value}"
    DISPLAY "The work/staff environment is {Env_value2}"
    DISPLAY "The Project Factor Correction is {PF}.";


RULE 1
IF DEVELOPER = END_USER
THEN
```

```
FIND END_USER_VALUE
PERSONNEL_PF =
(END_USER_VALUE*GT_TOOL_EXP_VALUE*NOVELTY_VALUE)
AF = (RELY*D_COMM*CHANGE*INTERFACE_COMPLX*OPER_EASE)
EV = (ENV_VALUE*ENV_VALUE2)
PF = (PERSONNEL_PF*METHOD_VALUE*AF*EV)
DISPLAY "The End-User value is {END_USER_VALUE}"
ELSE
FIND OVERALL_EXPVALUE
PERSONNEL_PF =
(OVERALL_EXPVALUE*GT_TOOL_EXP_VALUE*NOVELTY_VALUE)
AF = (RELY*D_COMM*CHANGE*INTERFACE_COMPLX*OPER_EASE)
EV = (ENV_VALUE*ENV_VALUE2)
PF = (PERSONNEL_PF*METHOD_VALUE*AF*EV)
DISPLAY "The Overall Experience value is {OVERALL_EXPVALUE}."

BECAUSE "If the software application is being developed by end
users it will generally take longer than if it is being developed
by professional DP staff";
```

```
RULE E_U_0
IF          END_USER_TYPE=comp_support_staff
THEN        END_USER_IMPACT=significant
BECAUSE     "Different types of end-users are skilled at different
levels. Some of them might be working as full-time computer
support staff and therefore represent professional IS personnel,
others may be functional support personnel, but some are basically
power-users and quite skilled. End-user programmers are relatively
less skilled, and the command level end-users are the least
skilled category (but still capable of developing simple 4GL
applications with help.)";
RULE E_U_1
IF          END_USER_TYPE=functional_support AND
            TRAINING=available AND
            INFO_CENTER_SUPPORT=available
THEN        END_USER_IMPACT=v_significant
BECAUSE     "End-users in this category are capable";

RULE E_U_2
IF          END_USER_TYPE=end_user_programmer AND
            TRAINING=available AND
```

```
          INFO_CENTER_SUPPORT=available
THEN      END_USER_IMPACT=v_v_significant
BECAUSE   "End-users in this category are inexperienced and need
          guidance from information center and good training";

RULE E_U_3
IF        END_USER_TYPE=command_level_user AND
          TRAINING=available AND
          INFO_CENTER_SUPPORT=available
THEN      END_USER_IMPACT=v_v_significant
BECAUSE   "End-users in this category are very inexperienced and
          need guidance from information center and good training";

RULE E_U_4
IF        END_USER_TYPE=functional_support AND
          TRAINING=not_available OR
          INFO_CENTER_SUPPORT=not_available
THEN      END_USER_IMPACT=v_v_significant
BECAUSE   "IF the end-users don't have the IC support and training
          it will affect their productivity";
```

```
RULE E_U_5

IF          END_USER_TYPE=end_user_programmer AND
            TRAINING=not_available OR
            INFO_CENTER_SUPPORT=not_available
THEN        END_USER_IMPACT=terminate
BECAUSE     "If this type of end-user does not have the IC support
            and training it will not be possible to develop any applications";


RULE E_U_6

IF          END_USER_TYPE=command_level_user AND
            TRAINING=not_available OR
            INFO_CENTER_SUPPORT=not_available
THEN        END_USER_IMPACT=terminate;


RULE E_U_7

IF          END_USER_IMPACT=SIGNIFICANT AND
            END_USER_SKILL=BASIC
THEN        END_USER_VALUE=1.42
BECAUSE     "End-users may range in experience by a fair degree. Some
            might simply have cursory experience with a 4GL product, others
```

```
        might have more significant experience. ";


RULE E_U_8
IF          END_USER_IMPACT=SIGNIFICANT AND
            END_USER_SKILL=COMFORTABLE_USE
THEN        END_USER_VALUE=1.21
BECAUSE     "Allocation of productivity rating";

RULE E_U_9
IF          END_USER_IMPACT=SIGNIFICANT AND
            END_USER_SKILL=AVERAGE
THEN        END_USER_VALUE=1.00
BECAUSE     "Allocation of productivity rating";

RULE E_U_10
IF          END_USER_IMPACT=SIGNIFICANT AND
            END_USER_SKILL=GOOD_PRACTICE
THEN        END_USER_VALUE=0.79
BECAUSE     "Allocation of productivity rating";
```

```
RULE E_U_11

IF          END_USER_IMPACT=SIGNIFICANT AND

            END_USER_SKILL=VERY_HIGH

THEN        END_USER_VALUE=0.58

BECAUSE     "Allocation of productivity rating";


RULE E_U_12

IF          END_USER_IMPACT=V_SIGNIFICANT AND

            END_USER_SKILL=BASIC

THEN        END_USER_VALUE=1.84

BECAUSE     "Allocation of productivity rating";


RULE E_U_13

IF          END_USER_IMPACT=V_SIGNIFICANT AND

            END_USER_SKILL=COMFORTABLE_USE

THEN        END_USER_VALUE=1.63

BECAUSE     "Allocation of productivity rating";


RULE E_U_14

IF          END_USER_IMPACT=V_SIGNIFICANT AND

            END_USER_SKILL=AVERAGE
```

```
THEN      END_USER_VALUE=1.42
BECAUSE   "Allocation of productivity rating";


RULE E_U_15
IF        END_USER_IMPACT=V_SIGNIFICANT AND
          END_USER_SKILL=GOOD_PRACTICE
THEN      END_USER_VALUE=1.21
BECAUSE   "Allocation of productivity rating";


RULE E_U_16
IF        END_USER_IMPACT=V_SIGNIFICANT AND
          END_USER_SKILL=VERY_HIGH
THEN      END_USER_VALUE=1.00
BECAUSE   "Allocation of productivity rating";


RULE E_U_17
IF        END_USER_IMPACT=V_V_SIGNIFICANT AND
          END_USER_SKILL=BASIC
THEN      END_USER_VALUE=2.47
BECAUSE   "Allocation of productivity rating";
```

```
RULE E_U_18
IF          END_USER_IMPACT=V_V_SIGNIFICANT AND
            END_USER_SKILL=COMFORTABLE_USE
THEN        END_USER_VALUE=2.26
BECAUSE     "Allocation of productivity rating";

RULE E_U_19
IF          END_USER_IMPACT=V_V_SIGNIFICANT AND
            END_USER_SKILL=AVERAGE
THEN        END_USER_VALUE=2.05
BECAUSE     "Allocation of productivity rating";

RULE E_U_20
IF          END_USER_IMPACT=V_V_SIGNIFICANT AND
            END_USER_SKILL=GOOD_PRACTICE
THEN        END_USER_VALUE=1.84
BECAUSE     "Allocation of productivity rating";

RULE E_U_21
IF          END_USER_IMPACT=V_V_SIGNIFICANT AND
            END_USER_SKILL=VERY_HIGH
```

```
THEN      END_USER_VALUE=1.63
BECAUSE   "Allocation of productivity rating";

RULE E_U_22
IF        END_USER_IMPACT=TERMINATE
THEN      END_USER_VALUE=0
DISPLAY   "It is not possible for this program to estimate a
          correction under such circumstances - program PF will be zero. "
BECAUSE   "It is difficult for the end-users to do any 4GL
          application development without access to some I.C. help.";

RULE 3
IF        OV_EXP_IMPACT=SIGNIFICANT AND
          OVERALL_EXPRATING=VERY_LOW
THEN      OVERALL_EXPVALUE=1.42
BECAUSE   "the overall experience of the developer can very
          significantly affect the speed and cost with which applications are
          developed.";

RULE 4
IF        OV_EXP_IMPACT=SIGNIFICANT AND
```

```
            OVERALL_EXPRATING=LOW
THEN        OVERALL_EXPVALUE=1.21
BECAUSE     "Allocation of productivity rating";

RULE 5
IF          OV_EXP_IMPACT=SIGNIFICANT AND
            OVERALL_EXPRATING=AVERAGE
THEN        OVERALL_EXPVALUE=1.0
BECAUSE     "Allocation of productivity rating";

RULE 6
IF          OV_EXP_IMPACT=SIGNIFICANT AND
            OVERALL_EXPRATING=HIGH
THEN        OVERALL_EXPVALUE=0.79
BECAUSE     "Allocation of productivity rating";

RULE 7
IF          OV_EXP_IMPACT=SIGNIFICANT AND
            OVERALL_EXPRATING=VERY_HIGH
THEN        OVERALL_EXPVALUE=0.58
BECAUSE     "Allocation of productivity rating";
```

RULE 8

IF GT_TOOL_EXP_IMPACT=SIGNIFICANT AND

GT_TOOL_EXP_RATING=VERY_LOW

THEN GT_TOOL_EXP_VALUE=1.28

BECAUSE "previous experience with a 4GL tool will significantly affect the efficiency and speed with which new applications can be developed. This can vary from very low (experience less than 1 month)

to very high (experience greater than 3 years). ";

RULE 9

IF GT_TOOL_EXP_IMPACT=SIGNIFICANT AND

GT_TOOL_EXP_RATING=LOW

THEN GT_TOOL_EXP_VALUE=1.14

BECAUSE "Allocation of productivity rating";

RULE 10

IF GT_TOOL_EXP_IMPACT=SIGNIFICANT AND

GT_TOOL_EXP_RATING=AVERAGE

THEN GT_TOOL_EXP_VALUE=1.00

BECAUSE "Allocation of productivity rating";

RULE 11

IF GT_TOOL_EXP_IMPACT=SIGNIFICANT AND

GT_TOOL_EXP_RATING=HIGH

THEN GT_TOOL_EXP_VALUE=0.86

BECAUSE "Allocation of productivity rating";

RULE 12

IF GT_TOOL_EXP_IMPACT=SIGNIFICANT AND

GT_TOOL_EXP_RATING=VERY_HIGH

THEN GT_TOOL_EXP_VALUE=0.72

BECAUSE "Allocation of productivity rating";

RULE 13

IF NOVELTY_IMPACT=SIGNIFICANT AND

NOVELTY_RATING=VERY_LOW

THEN NOVELTY_VALUE=1.28

BECAUSE "previous experience with a similar project and application can affect the cost of development. Novel applications, which the development team has no previous familiarity with, should require more manpower. By the same token, if a project is simple (and not unique) and if it lends itself to automatic application generation, it's novelty is classified as very low. ";

```
RULE 14
IF          NOVELTY_IMPACT=SIGNIFICANT AND
            NOVELTY_RATING=LOW
THEN        NOVELTY_VALUE=1.14
BECAUSE     "Allocation of productivity rating";

RULE 15
IF          NOVELTY_IMPACT=SIGNIFICANT AND
            NOVELTY_RATING=AVERAGE
THEN        NOVELTY_VALUE=1.00
BECAUSE     "Allocation of productivity rating";

RULE 16
IF          NOVELTY_IMPACT=SIGNIFICANT AND
            NOVELTY_RATING=HIGH
THEN        NOVELTY_VALUE=0.86
BECAUSE     "Allocation of productivity rating";

RULE 17
```

```
IF          NOVELTY_IMPACT=SIGNIFICANT AND
            NOVELTY_RATING=VERY_HIGH
THEN        NOVELTY_VALUE=0.72
BECAUSE     "Allocation of productivity rating";


RULE MTHD_0
IF          TECHNIQUE=jad
THEN        TECHNIQUE_EFFECT=good;


RULE MTHD_1
IF          TECHNIQUE=prototyping
THEN        TECHNIQUE_EFFECT=good;


RULE MTHD_2
IF          TECHNIQUE=jad_and_prototyping
THEN        TECHNIQUE_EFFECT=good;


RULE MTHD_3A
IF          TECHNIQUE=PF_has_no_influence
THEN        METHOD_VALUE = 1.0
BECAUSE     "Techniques such as JAD - joint application design and
```

```
             development and subsequent prototyping serve to reduce to
             development costs.";

RULE MTHD_3B
IF           PRACTICE = PF_has_no_influence
THEN         METHOD_VALUE = 1.0
BECAUSE      "Techniques such as top down design, structured design and
             programming, and related strategies results in an orderly
             development of the software, moreover, it assures that there will
             be no chaos during development.";


RULE MTHD_4
IF           PRACTICE=struct_design_&_prog
THEN         PRACTICE_EFFECT=good;

RULE MTHD_5
IF           PRACTICE=walkthroughs
THEN         PRACTICE_EFFECT=good;

RULE MTHD_6
```

```
IF          PRACTICE=struct_tech_&_walkth
THEN        PRACTICE_EFFECT=good;

RULE MTHD_8
IF          TECHNIQUE_EFFECT=good AND
            PRACTICE_EFFECT=good
THEN        IMPACT=significant;

RULE MTHD_9
IF          TECHNIQUE_EFFECT=PF_has_no_influence AND
            PRACTICE_EFFECT=good
THEN        IMPACT=average;

RULE MTHD_10
IF          TECHNIQUE_EFFECT=good AND
            PRACTICE_EFFECT=PF_has_no_influence
THEN        IMPACT=average;

RULE MTHD_11
IF          IMPACT=significant AND
            METHOD_EXPERIENCE=very_low
```

```
THEN      METHOD_VALUE=1.00
BECAUSE   "while end-users or developers might be enthusiastic about
          using techniques such as JAD/Prototyping or structured methods
for
          software development, they may not be very experienced with such
          techniques.";


RULE MTHD_12
IF        IMPACT=significant AND
          METHOD_EXPERIENCE=low
THEN      METHOD_VALUE=1.00;

RULE MTHD_13
IF        IMPACT=significant AND
          METHOD_EXPERIENCE=average
THEN      METHOD_VALUE=1.00;

RULE MTHD_14
IF        IMPACT=significant AND
          METHOD_EXPERIENCE=high
```

```
THEN      METHOD_VALUE=0.86;

RULE MTHD_15
IF        IMPACT=significant AND
          METHOD_EXPERIENCE=very_high
THEN      METHOD_VALUE=0.72;

RULE MTHD_16
IF        IMPACT=average AND
          METHOD_EXPERIENCE=very_low
THEN      METHOD_VALUE=1.14;

RULE MTHD_17
IF        IMPACT=average AND
          METHOD_EXPERIENCE=low
THEN      METHOD_VALUE=1.07;

RULE MTHD_18
IF        IMPACT=average AND
          METHOD_EXPERIENCE=average
THEN      METHOD_VALUE=1.00;
```

```
RULE MTHD_19
IF          IMPACT=average AND
            METHOD_EXPERIENCE=high
THEN        METHOD_VALUE=1.00;


RULE MTHD_20
IF          IMPACT=average AND
            METHOD_EXPERIENCE=very_high
THEN        METHOD_VALUE=1.00;


RULE O_EASE_0
IF          OPERATIONAL_EASE=very_low
THEN        OPER_EASE=0.72
BECAUSE     "If ease of operating the system is essential then the
            application must be automated to a great degree, for instance, this
            might involve special effort to provide error recovery, automatic
            backup of database files, and easy start-up, or shut-down.";


RULE O_EASE_1
IF          OPERATIONAL_EASE=low
THEN        OPER_EASE=0.86;
```

RULE O_EASE_2

IF OPERATIONAL_EASE=average

THEN OPER_EASE=1.00;

RULE O_EASE_3

IF OPERATIONAL_EASE=high

THEN OPER_EASE=1.14;

RULE O_EASE_4

IF OPERATIONAL_EASE=very_high

THEN OPER_EASE=1.28;

RULE O_EASE_5

IF OPERATIONAL_EASE=PF_not_applicable

THEN OPER_EASE=1.00;

RULE I_CMPLX_0

IF INTERFACE_COMPLEXITY=very_low

THEN INTERFACE_COMPLX=0.72

BECAUSE "On-line functions for data entry, update, or output involve more effort than similar batch functions. If some special interfaces are required such as output to a laser printer or graphics terminal more effort is required. ";

```
RULE I_CMPLX_1
IF          INTERFACE_COMPLEXITY=low
THEN        INTERFACE_COMPLX=0.86;

RULE I_CMPLX_2
IF          INTERFACE_COMPLEXITY=average
THEN        INTERFACE_COMPLX=1.00;

RULE I_CMPLX_3
IF          INTERFACE_COMPLEXITY=high
THEN        INTERFACE_COMPLX=1.14;

RULE I_CMPLX_4
IF          INTERFACE_COMPLEXITY=very_high
THEN        INTERFACE_COMPLX=1.28;

RULE I_CMPLX_5
IF          INTERFACE_COMPLEXITY=PF_not_applicable
```

```
THEN       INTERFACE_COMPLX=1.00;

RULE C0
IF         RELIABILITY_RATING=very_low
THEN       RELY=0.72
BECAUSE    "Reliability expected from the application being developed
           can significantly affect the cost of application development. An
           application must be classified as requiring high reliability if
           failure can result in a large financial losses";

RULE C1
IF         RELIABILITY_RATING=low
THEN       RELY=0.86;

RULE C2
IF         RELIABILITY_RATING=average
THEN       RELY=1.00;

RULE C3
IF         RELIABILITY_RATING=high
THEN       RELY=1.14;
```

```
RULE C4
IF        RELIABILITY_RATING=very_high
THEN      RELY=1.28;

RULE C5
IF        RELIABILITY_RATING=PF_not_applicable
THEN      RELY=1.00;

RULE COMM_0
IF        DATA_COMMUNICATION=very_low
THEN      D_COMM=0.72
BECAUSE   "More effort is required to develop applications that
          require data to be sent or received over multiplexers, networks, or
          LANs.";

RULE COMM_1
IF        DATA_COMMUNICATION=low
THEN      D_COMM=0.86;

RULE COMM_2
IF        DATA_COMMUNICATION=average
```

```
THEN        D_COMM=1.00;


RULE COMM_3
IF          DATA_COMMUNICATION=high
THEN        D_COMM=1.14;


RULE COMM_4
IF          DATA_COMMUNICATION=very_high
THEN        D_COMM=1.28;


RULE COMM_5
IF          DATA_COMMUNICATION=PF_not_applicable
THEN        D_COMM=1.00;


RULE F_C_0
IF          FACILITATE=very_low
THEN        CHANGE=0.72
BECAUSE     "if the application is designed to facilitate changes
            (such as frequent updates of tax rates, or interest rates), or if
            it is designed to facilitate maintenance or creation of module
            libraries more effort will be required. ";
```

```
RULE F_C_1
IF          FACILITATE=low
THEN        CHANGE=0.86;

RULE F_C_2
IF          FACILITATE=average
THEN        CHANGE=1.00;

RULE F_C_3
IF          FACILITATE=high
THEN        CHANGE=1.14;

RULE F_C_4
IF          FACILITATE=very_high
THEN        CHANGE=1.28;

RULE F_C_5
IF          FACILITATE=PF_not_applicable
THEN        CHANGE=1.00;
```

```
RULE ENV_1
IF          ENVIRONMENT=Xeroxing_&_Printing AND
            ENVIRONMENT=Workstations AND
            ENVIRONMENT=Technical_Education AND
            ENVIRONMENT=Software_&_Hardware
THEN        ENV_RATING=average
BECAUSE     "attributes such as availability of xeroxing and
            duplicating resources, private work station, availability of all
            required software and hardware needed for the project, and
            technical training (if needed) will affect the project estimates";

RULE ENV_2
IF          ENVIRONMENT=Xeroxing_&_Printing AND
            ENVIRONMENT=Workstations AND
            ENVIRONMENT=Technical_Education
THEN        ENV_RATING=v_low;


RULE ENV_3
IF          ENVIRONMENT=Workstations AND
            ENVIRONMENT=Technical_Education AND
            ENVIRONMENT=Software_&_Hardware
THEN        ENV_RATING=average;


RULE ENV_4
IF          ENVIRONMENT=Xeroxing_&_Printing AND
            ENVIRONMENT=Technical_Education AND
            ENVIRONMENT=Software_&_Hardware
THEN        ENV_RATING=average;


RULE ENV_5
IF          ENVIRONMENT=Xeroxing_&_Printing AND
            ENVIRONMENT=Workstations
THEN        ENV_RATING=low;


RULE ENV_6
IF          ENVIRONMENT=Workstations AND
            ENVIRONMENT=Technical_Education
THEN        ENV_RATING=v_low;


RULE ENV_7
IF          ENVIRONMENT=Technical_Education AND
```

```
                ENVIRONMENT=Software_&_Hardware
THEN            ENV_RATING=low;


RULE ENV_8
IF              ENVIRONMENT=Xeroxing_&_Printing AND
                ENVIRONMENT=Technical_Education
THEN            ENV_RATING=low;


RULE ENV_9
IF              ENVIRONMENT=Xeroxing_&_Printing AND
                ENVIRONMENT=Software_&_Hardware
THEN            ENV_RATING=low;


RULE ENV_10
IF              ENVIRONMENT=Workstations AND
                ENVIRONMENT=Software_&_Hardware
THEN            ENV_RATING=low;


RULE ENV_11
IF              ENVIRONMENT=Xeroxing_&_Printing
THEN            ENV_RATING=v_low;
```

```
RULE ENV_12

IF          ENVIRONMENT=Workstations

THEN        ENV_RATING=v_low;


RULE ENV_13

IF          ENVIRONMENT=Technical_Education

THEN        ENV_RATING=v_low;


RULE ENV_14

IF          ENVIRONMENT=Software_&_Hardware

THEN        ENV_RATING=v_low;


RULE ENV_15

IF          ENVIRONMENT=PF_not_applicable

THEN        ENV_RATING =average;


RULE ENV_16

IF          env_rating=v_low

THEN        env_value=1.14;


RULE ENV_17
```

```
IF          env_rating=low
THEN        env_value=1.07;


RULE ENV_18
IF          env_rating = average
THEN        env_value = 1.00;


RULE ENV_19
IF          Environment2=PF_not_applicable
THEN        ENV_value =1.00;



RULE ENV2_1
IF          Environment2=Inadequate_compensation AND
            Environment2=Low_priority_projs AND
            Environment2=Many_concurrent_proj AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=poor

BECAUSE     "inadequate compensation or low morale of employee could
            affect the productivity. If the developers are working on a
```

```
        project that has low visibility and priority, or if they are
        working on several projects at the same time this will also affect
        the cost estimates";
RULE ENV2_2
IF          Environment2=Inadequate_compensation AND
            Environment2=Low_priority_projs AND
            Environment2=Many_concurrent_proj
THEN        ENV_RATING2=v_low;

RULE ENV2_3
IF          Environment2=Low_priority_projs AND
            Environment2=Many_concurrent_proj AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=average;

RULE ENV2_4
IF          Environment2=Inadequate_compensation AND
            Environment2=Many_concurrent_proj AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=average;
```

```
RULE ENV2_5
IF          Environment2=Inadequate_compensation AND
            Environment2=Low_priority_projs
THEN        ENV_RATING2=low;

RULE ENV2_6
IF          Environment2=Low_priority_projs AND
            Environment2=Many_concurrent_proj
THEN        ENV_RATING2=v_low;

RULE ENV2_7
IF          Environment2=Many_concurrent_proj AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=low;

RULE ENV2_8
IF          Environment2=Inadequate_compensation AND
            Environment2=Many_concurrent_proj
THEN        ENV_RATING2=low;

RULE ENV2_9
```

```
IF          Environment2=Inadequate_compensation AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=low;


RULE ENV2_10
IF          Environment2=Low_priority_projs AND
            Environment2=Morale_not_high
THEN        ENV_RATING2=low;


RULE ENV2_11
IF          Environment2=Inadequate_compensation
THEN        ENV_RATING2=v_low;


RULE ENV2_12
IF          Environment2=Low_priority_projs
THEN        ENV_RATING2=v_low;


RULE ENV2_13
IF          Environment2=Many_concurrent_proj
THEN        ENV_RATING2=v_low;
```

```
RULE ENV2_14
IF          Environment2=Morale_not_high
THEN        ENV_RATING2=v_low;

RULE ENV2_15
IF          Environment2=PF_not_applicable
THEN        ENV_value2 =1.00
BECAUSE     "inadequate compensation or low morale of employees could
            affect their productivity. If the developers are working on a
            project that has low visibility and priority, or if they are
            working on several projects at the same time it will also affect
            the cost estimates";

RULE ENV2_16
IF          env_rating2=v_low
THEN        env_value2=1.07;

RULE ENV2_17
IF          env_rating2=low
THEN        env_value2=1.14;
```

```
RULE ENV2_18
IF          env_rating2 = poor
THEN        env_value2 = 1.21;


ASK DEVELOPER: "Who is going to develop the application?";
CHOICES DEVELOPER: END_USER, DP_STAFF;


ASK OVERALL_EXPRATING: "What is the overall EXPERIENCE of the
development team?";
CHOICES OVERALL_EXPRATING:
VERY_LOW,LOW,AVERAGE,HIGH,VERY_HIGH;


ASK 4GT_TOOL: "How much familiarity do the developers have with the
DBMS and
Fourth Generation Tools to be used in implementing the
application?";
CHOICES 4GT_TOOL: VERY_LOW, LOW, AVERAGE, HIGH,
VERY_HIGH;


ASK GT_TOOL_EXP_RATING: "What is the level of expertise with the
4GT TOOL?";
```

CHOICES GT_TOOL_EXP_RATING:

VERY_LOW,LOW,AVERAGE,HIGH,VERY_HIGH;

ASK NOVELTY_RATING: "How familiar is the project team with the with the software development project on hand?";

CHOICES NOVELTY_RATING:

VERY_LOW,LOW,AVERAGE,HIGH,VERY_HIGH;

ASK END_USER_SKILL: "What is the level of end-user computing skills?";

CHOICES END_USER_SKILL:

BASIC,COMFORTABLE_USE,AVERAGE,GOOD_PRACTICE,VERY_HIGH;

ASK END_USER_TYPE: "What is the end-user type?";

CHOICES END_USER_TYPE:

comp_support_staff,functional_support,

end_user_programmer,command_level_user;

ASK TRAINING: "Is training available?";

CHOICES TRAINING: available,not_available;

ASK INFO_CENTER_SUPPORT: "Is Information Center support available readily?";

CHOICES INFO_CENTER_SUPPORT: available,not_available;

ASK TECHNIQUE: "What software development techniques will be used?";

CHOICES TECHNIQUE:

jad,prototyping,jad_and_prototyping,PF_has_no_influence;

ASK PRACTICE: "What software implementation techniques will be used?";

CHOICES PRACTICE:

struct_design_&_prog,walkthroughs,struct_tech_&_walkth,

PF_has_no_influence;

ASK METHOD_EXPERIENCE: "What is the level of experience with techniques such as JAD, and structured programming?";

CHOICES METHOD_EXPERIENCE: very_low,low,average,high,very_high;

ASK RELIABILITY_RATING: "How important is software reliability?";

CHOICES RELIABILITY_RATING:

```
very_low,low,average,high,very_high,PF_not_applicable;

ASK DATA_COMMUNICATION: "Is data communication crucial here?";
CHOICES DATA_COMMUNICATION:
very_low,low,average,high,very_high,PF_not_applicable;

ASK FACILITATE: "Is the application designed to facilitate
change?";
CHOICES FACILITATE:
very_low,low,average,high,very_high,PF_not_applicable;

ASK INTERFACE_COMPLEXITY: "How critical is the interface
complexity?";
CHOICES INTERFACE_COMPLEXITY:
very_low,low,average,high,very_high,PF_not_applicable;

ASK OPERATIONAL_EASE: "Should the software provide a degree of
operational ease?";
CHOICES OPERATIONAL_EASE:
very_low,low,average,high,very_high,PF_not_applicable;
```

ASK ENVIRONMENT: "Which of the following facilities are very adequately available in the project environment (select all that apply)?";

CHOICES ENVIRONMENT:

Xeroxing_&_Printing,Workstations,Technical_Education,

Software_&_Hardware, PF_not_applicable;

ASK Environment2: "Select all factors that describe the environment that the staff are working in?";

CHOICES Environment2:

Inadequate_Compensation,Low_Priority_Projs,Many_Concurrent_Proj,

Morale_Not_High, PF_Not_Applicable;

PLURAL: ENVIRONMENT;

PLURAL: Environment2;

Appendix B

User Interface: Questions asked by PFES

- Who is going to develop the application?

  end-user, dp staff

- What is the overall experience of the development team?

  very low, low, average, high, very high

- How much familiarity do the developers have with the DBMS and Fourth Generation Tools to be used in implementing the application?

  very low, low, average, high, very high

- What is the level of expertise with the 4GT TOOL?

  very low, low, average, high, very high

- How familiar is the project team with the with the software development project on hand?

  very low, low, average, high, very high

- What is the level of end-user computing skills?

basic, comfortable use, average, good practice, very high

- What is the end-user type?

  comp support staff, functional support,

  end-user programmer, command level user

- Is training available?

  available, not available

- Is Information Center support available readily?

  available, not available

- What software development techniques will be used?

  jad, prototyping, jad and prototyping, PF has no influence

- What software implementation techniques will be used?

  struct design & prog, walkthroughs, struct tech & walkth,

  PF has no influence

- What is the level of experience with techniques such as JAD, and structured programming?

very low, low, average, high, very high

- How important is software reliability?

  very low, low, average, high, very high, PF not applicable

- Is data communication crucial here?

  very low, low, average, high, very high, PF not applicable

- Is the application designed to facilitate change?

  very low, low, average, high, very high, PF not applicable

- How critical is the interface complexity?

  very low, low, average, high, very high, PF not applicable

- Should the software provide a degree of operational ease?

  very low, low, average, high, very high, PF not applicable

- Which of the following facilities are very adequately available in the project environment (select all that apply)?

  Xeroxing & Printing, Workstations, Technical Education, Software & Hardware, PF not applicable o Select all factors that describe the environment that the staff are working in?

Inadequate Compensation, Low Priority Projs, Many Concurrent Proj, Morale Not High, PF Not Applicable

Appendix C

PFES S.S. Model Version

Project Factor	Enter Choice (0-5)	Options	(Enter 0,1,2,3,4, or 5)				
Type of Developer	2	1. End User	2. DP Staff				
End User Type	0	1. CSS	2. FS	3. EUP	4. CL		
End User Skill Level	0	1. Novice	2.Comfor -table	3. Average	4. Good	5.V.Good	
End User Resources	0	1. Training Available	2. Training Unavailable				
Information Center Support	0	1. Support Available	2. Support Unavailable				
DP Staff Experience Level	3	1. V. Low	2. Low	3. Average	4. High	5. V. High	
Expertise with 4GTs	3	1. V. Low	2. Low	3. Average	4. High	5. V. High	
Familiarity with Similar Project	3	1. V. Low	2. Low	3. Average	4. High	5. V. High	
Data Communication Complexity	3	1. V. Low	2. Low	3. Average	4. High	5. V.High	0. Not Valid
Inteface Complexity	3	1. V. Low	2. Low	3. Average	4. High	5.V. High	0. Not Valid
Software Reliability	3	1. V. Low	2. Low	3. Average	4. High	5. V.High	0. Not Valid
Operational Ease	3	1. V. Low	2. Low	3. Average	4. High	5. V.High	0. Not Valid
Environment	3	1. V. Low	2. Low	3. Average	4. High	5. V.High	0. Not Valid
Familiarity with SDLC	3	1. V. Low	2. Low	3. Average	4. High	5. V.High	0. Not Valid
Recommended PFES Correction:	1.0						
Note: Select 0, if not valid or if manually coded; as with Process Functions							

Appendix D

4GT Model Program Code

```
A1: [W19] '4GT MODEL - Calibrated For LEGASY

A2: [W19] 'Forms

D2: (F2) (D15)

E2: [W12] 'person-hours

A3: [W19] 'Reports

D3: (F2) (D21)

E3: [W12] 'person-hours

A4: [W19] 'Data Type

D4: (F2) (D25)

E4: [W12] 'person-hours

A5: [W19] 'Process Type

D5: (F2) 125

E5: [W12] 'person-hours

A6: [W19] 'Development Effort (estimated)

D6: (F2) @SUM(D5..D2)

E6: [W12] 'person-hours

A7: [W19] 'Development Effort (actual)
```

D7: (F2) 2340

E7: [W12] 'person-hours

A8: [W19] 'Expansion Factor (actual/estimated)

D8: (F2) (D7/D6)

A9: [W19] 'F O R M S

A10: [W19] 'SE Category

B10: 'SE Value

C10: [W11] 'Magnitude

D10: 'Total SE Value

A11: [W19] 'Simple SE

B11: (F2) 0.13

C11: [W11] 185

D11: (C11*B11)

A12: [W19] 'Basic SE

B12: (F2) 0.29

C12: [W11] 36

D12: (C12*B12)

A13: [W19] 'Detailed SE

B13: (F2) 1.59

C13: [W11] 32

D13: (C13*B13)

```
A14: [W19] 'User Exit

B14: (F2) 22.72

C14: [W11] 11

D14: (F2) (C14*B14)

A15: [W19] 'Total Effort

D15: @SUM(D14..D11)

E15: [W12] 'person-hours

A16: [W19] 'R E P O R T S

A17: [W19] 'SE Category

B17: 'SE Value

C17: [W11] 'Magnitude

D17: 'Total SE Value

A18: [W19] 'Simple SE

B18: (F2) 0.13

C18: [W11] 75

D18: (F2) (C18*B18)

A19: [W19] 'Basic SE

B19: (F2) 0.84

C19: [W11] 14

D19: (F2) (C19*B19)

A20: [W19] 'Detailed SE
```

```
B20: (F2) 2.55

C20: [W11] 69

D20: (F2) (C20*B20)

A21: [W19] 'Total Effort

D21: @SUM(D20..D18)

E21: [W12] 'person-hours

A22: [W19] 'D A T A

A23: [W19] 'SE Category

B23: 'SE Value

C23: [W11] 'Magnitude

D23: 'Total SE Value

A24: [W19] 'Data Element

B24: (F2) 0.41

C24: [W11] 238

D24: (F2) (C24*B24)

A25: [W19] 'Total Effort

D25: +D24

E25: [W12] 'person-hours
```

```
************************************************************
*-- Name....: model.prg
*-- Notes...: main program, provides pop-up menu for selecting
              specification elements.
************************************************************
SET TALK OFF
set stat off
close all
clear
*
select 1
use data1
append blank
@ 1,0 to 12,60 double
@ 2, 18 say " FUNCTION DEFINITION"
@ 3, 1 to 3, 59
@ 5, 10 say "Function Name: " get name1
@ 7, 10 say "Function Type: " get type
@ 9, 10 say "SEV: "
@ 9, 40 say "Level :"
read
```

```
 if len(trim(type)) = 0
    sele 3
    use data3
    define popup rob from 12, 5 to 19, 25 prompt field vkey
    on selection popup rob deactivate popup
    activate popup rob
    goto bar()
    replace data1->type with data3->vkey
 endif
 @ 7, 10 say "Function Type: " get data1->type
 clear gets
 *
 sele 1
 @ 9, 10 say "SEV: " get name2
 do morepop
 close all
 deactivate window rob1
 set talk on
 set stat on
 return
 Procedure morepop
 *
 * if the field <number> has been filled, then ignore the popup window.
 *
   select 2
   use data2
   * Define the popup window for input
   *
   define window rob1 from 11, 10 to 17, 40 double
   store 0 to vtotal
   do while .t.
     brows lock 0  noappend noedit noclear nodelete nomenu window rob1
     read


     if readkey() = 12
        deactivate window rob1
        return
     endif
     do adding
  enddo
return
```

```
*
procedure adding
*
*  Sum up one or more record from one database on a single field
*
   sele 4
   use data4
   define popup rob4 from 11, 60 to 17, 70 prompt field level
   on selection popup rob4 do selecting
   activate popup rob4


   select 2
   vtotal = vtotal + number
   sele 1        && selecting where to put the calculated value
   replace data1->name2 with vtotal
   @ 9, 10 say "SEV: " get data1->name2
   sele 2        && return to source file
return


Procedure selecting
  if bar() > 0
```

```
        goto bar()
        replace data1->vlevel with level
        @ 9, 40 say "Level : " get level
        deactivate popup
        sele 2
    endif
return

************************************************************
*-- Name....: FRM_TYPE.FMT
*-- Version.: dBASE IV, Format 1.0
*-- Notes...: Format files use "" as delimiters!
************************************************************

*-- Format file initialization code -----------------------------------------

IF SET("TALK")="ON"
   SET TALK OFF
   lc_talk="ON"
ELSE
   lc_talk="OFF"
```

```
ENDIF


*-- This form was created in MONO mode

SET DISPLAY TO MONO


lc_status=SET("STATUS")

*-- SET STATUS was ON when you went into the Forms Designer.

IF lc_status = "OFF"

  SET STATUS ON

ENDIF



*-- @ SAY GETS Processing. -----------------------------------------------


*--  Format Page: 1


@ 3,0 SAY "             Specification Element: "

@ 3,37 GET name1 PICTURE

"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

@ 5,0 SAY "             Total Specification Element Value: "

@ 5,49 GET name2 PICTURE "999999"
@ 7,0 SAY "              Function Type:     "
@ 7,34 GET type PICTURE "XXXXXXXXXXXXX"
@ 9,0 SAY "              Average Magnitude:  "
@ 9,34 GET vlevel PICTURE "XXXXX"


*-- Format file exit code -----------------------------------------------


*-- SET STATUS was ON when you went into the Forms Designer.
IF lc_status = "OFF"  && Entered form with status off
   SET STATUS OFF     && Turn STATUS "OFF" on the way out
ENDIF


IF lc_talk="ON"
   SET TALK ON
ENDIF
RELEASE lc_talk,lc_fields,lc_status
*-- EOP: FRM_TYPE.FMT
```

Appendix E

Functions Involved With Calibration

LEGASY:

Calibration of functions is described in the next appendix. Some sample screens associated with the process are illustrated here. Due to proprietary restrictions only a selected few are described in detail here.

```
                                    Litigation

File Name                            OpenDate   Type Litg IHC OC  Dept Prov Resl
¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ ¤¤/¤¤/¤¤ ¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤ ¤¤¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤¤


Service Date ¤¤/¤¤/¤¤                            Amount at Issue $ ¤¤,¤¤¤,¤¤¤.¤¤

    Province ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤         Amount Claimed $ ¤¤,¤¤¤,¤¤¤.¤¤

         City ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤          Punitive Damages $ ¤¤,¤¤¤,¤¤¤.¤¤

Judicial District ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤       General Damages $ ¤¤,¤¤¤,¤¤¤.¤¤

        Court ¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤         TOTAL Claimed $¤¤¤,¤¤¤,¤¤¤.¤¤

Comment ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
        ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
        ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                                                              1a. litigation.scr
```

- This screen is used to add/modify/query Litigation information related to a specific case.

Possible Queries:
- display litigation information for cases in 'B.C.' and Amount at Issues > $250,000.
- scroll through all cases where Court is 'Queens Bench'

```
                                File Information                    ID # ¤¤¤¤¤¤

  File Name ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤      Opened Date ¤¤/¤¤/¤¤

 File Type                    Litigation Type            Amt at Issue  Status
 ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤    ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  $¤¤,¤¤¤,¤¤¤.¤¤      ¤

Major Issues ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤         ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
 Case Status ¤¤/¤¤/¤¤  ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
   Next Step ¤¤/¤¤/¤¤  ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

In-house (y/n) ¤                    Province ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

IHC ¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  OC Firm ¤¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
    ¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  OC Name ¤¤¤¤  ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
Time    ¤¤,¤¤¤.¤                Current Year ¤,¤¤¤,¤¤¤.¤¤ Case Total ¤,¤¤¤,¤¤¤.¤¤
                                             Phone #  Line Department
Contact ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ ¤¤¤¤¤¤¤¤ ¤¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

 Case Resolution ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ Closed Date ¤¤/¤¤/¤¤

   Old File Type ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤ Change Date ¤¤/¤¤/¤¤     NotePad? ¤
                                                                  1.  file.scr
```

- this screen should be used to open a new File, display or make changes to an exiting file
- the ID # will be assigned by the system at the time the file is opened
- a check list may be displayed reminding the operator of other task which should be performed when a file is opened
- the following fields may be "looked up" in a list of available values:
  - File Type
  - Litigation Type
  - Status
  - Province
  - IHC
  - OC Firm
  - OC Name
  - Contact Person
  - Line Department
  - Case Resolution Code
- Litigation Type should only be entered when the File Type is 'Litigation'
- the operator may query on all fields that may be entered
- the operator may "count query hits" and "scroll through" query results

```
                                   Facts

File Name                                 OpenDate    Type Litg IHC OC  Dept Prov Resl
¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  ¤¤/¤¤/¤¤ ¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤ ¤¤¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤¤

     Employer ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤        Job Description (y/n) ¤

      Employment Commenced ¤¤/¤¤/¤¤          Elimination Period from ¤¤/¤¤/¤¤
                Terminated ¤¤/¤¤/¤¤                               to ¤¤/¤¤/¤¤
          Last Date Worked ¤¤/¤¤/¤¤          Benefits   (y/n) ¤  from ¤¤/¤¤/¤¤
Returned to Work  (y/n) ¤  ¤¤/¤¤/¤¤                               to ¤¤/¤¤/¤¤

                                    Periodic AMT $¤¤¤,¤¤¤.¤¤ / ¤¤¤¤¤¤¤¤¤¤

        Cause of Disability ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
     Disability / Diagnosis ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                 WCB (y/n) ¤  Comment ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
                 CPP (y/n) ¤          ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
                 UIC (y/n) ¤          ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                                                               1b.  facts.scr
```

- This screen is used to add/modify/query Facts related to a specific case.

```
                      Plaintiff and Policy Information

File Name                                 OpenDate    Type Litg IHC OC  Dept Prov Resl
¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  ¤¤/¤¤/¤¤ ¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤ ¤¤¤ ¤¤¤¤ ¤¤¤¤ ¤¤¤¤

              Plaintiff Name ¤¤¤¤ ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                   Date of Birth ¤¤/¤¤/¤¤     Age ¤¤       Sex ¤

                       Education ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                       Ocupation ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

    Policy # ¤¤¤¤¤¤¤¤¤¤              Policyholder ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

 Policy Type ¤¤¤  ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤         Agent ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

   Policy Effective Date ¤¤/¤¤/¤¤

                                                               1c.  policy.scr
```

- This screen is used to add/modify/query Policy and Plaintiff information related to a specific case.

```
                                   Office

Code ¤¤¤¤¤        Name ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤

               Address ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
                       ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
                  City ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
              Province ¤¤¤¤         Postal Code ¤¤¤ ¤¤¤

               Phone # ¤¤¤¤¤¤¤¤¤¤¤¤¤¤

                 Fax # ¤¤¤¤¤¤¤¤¤¤¤¤¤¤
```

13. office.lup

- The purpose of this screen is to add/modify/query on information for a specific office, such as Outside Counsel Office.
- The information on this screen should exist prior to setting up any Parties from an associated office.

Possible Queries:
- list all offices in 'Alberta'

Appendix F

Identification of Form SEs

FORMS										
Name	CASL	CITL	COUL	DIVL	FTYL	ISSL	LITL	OFFL	PARL	PCYL
Simple SE	2	1	2	2	2	1	2	2	4	2
Basic SE	0	0	0	0	0	0	0	0	0	0
Detailed SE	1	1	1	1	1	1	1	1	1	1
User Exit	0	0	0	0	0	0	0	0	0	0
Name	PROL	PYTL	PYML	TASL	FISA	ISSQ	FHD	SLDQ	FEEA	LFFQ
Simple SE	2	2	3	2	5	13	7	7	13	17
Basic SE	0	0	0	0	0	0	4	2	9	6
Detailed SE	1	1	1	1	1	2	2	0	3	1
User Exit	0	0	0	0	1	2	0	0	0	2
Name	FPAA	PARQ	SLTA	FTTQ	SLTQ	PARA	FTYQ	OCH	Total	Avg SEVs
Simple SE	5	18	9	7	13	25	8	9	185	0.13
Basic SE	0	1	5	1	2	4	0	0	36	0.29
Detailed SE	0	0	2	0	1	5	0	1	32	1.59
User Exit	2	2	0	0	2	0	0	0	11	22.73

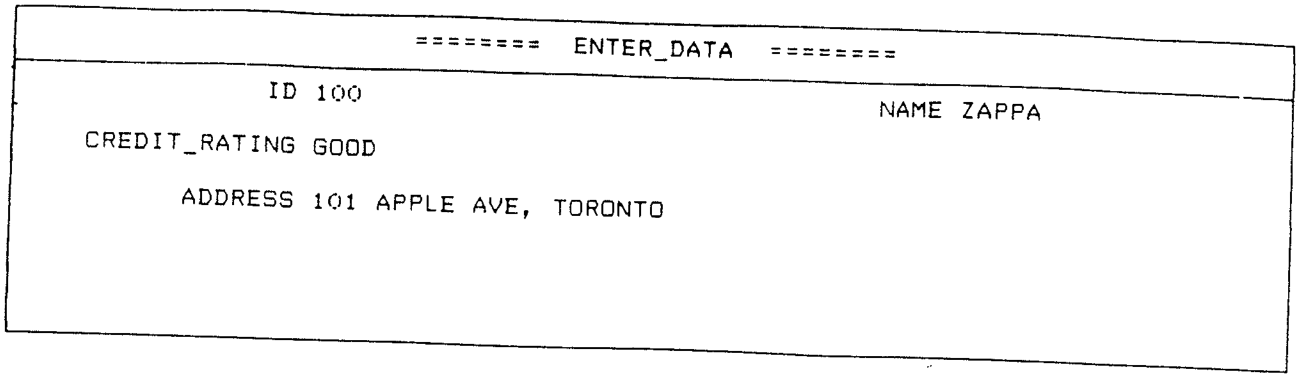
Appendix G

Identification of Report SEs

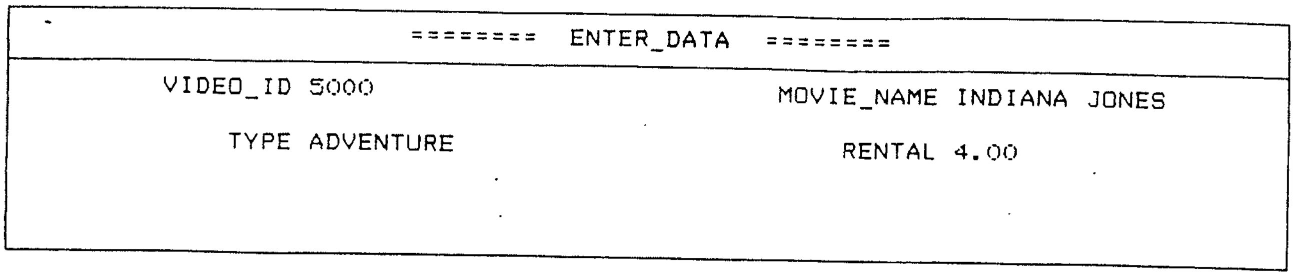
REPORTS								
Name	QLA	QLB	LFR	CLCL	NCLD	CHIH	JD	PL
Screen Field SE	0	0	0	0	0	0	1	8
Basic SE	0	0	0	4	0	0	1	1
Detailed SE	1	1	4	0	7	7	0	0
Name	NCBP	NNCEM	NGDCBY	OCCAFS	CCFY	NNCQ	OL	LDL
Screen Field SE	0	0	0	0	0	0	10	120 5
Basic SE	0	0	0	0	0	0	1	1
Detailed SE	7	20	6	2	1	6	0	0
Name	CD	LR1	LR2	TL	RC	ISS.	LT	XT
Screen Field SE	0	15	12	3	2	1	2	16
Basic SE	0	1	1	1	1	1	1	0
Detailed SE	6	0	0	0	0	0	0	1
Total SF = 75 AVG SEV = 0.13	Total Basic = 14 AVG SEV = 0.84		Total Detail = 69; AVG SEV = 2.55					

Appendix H

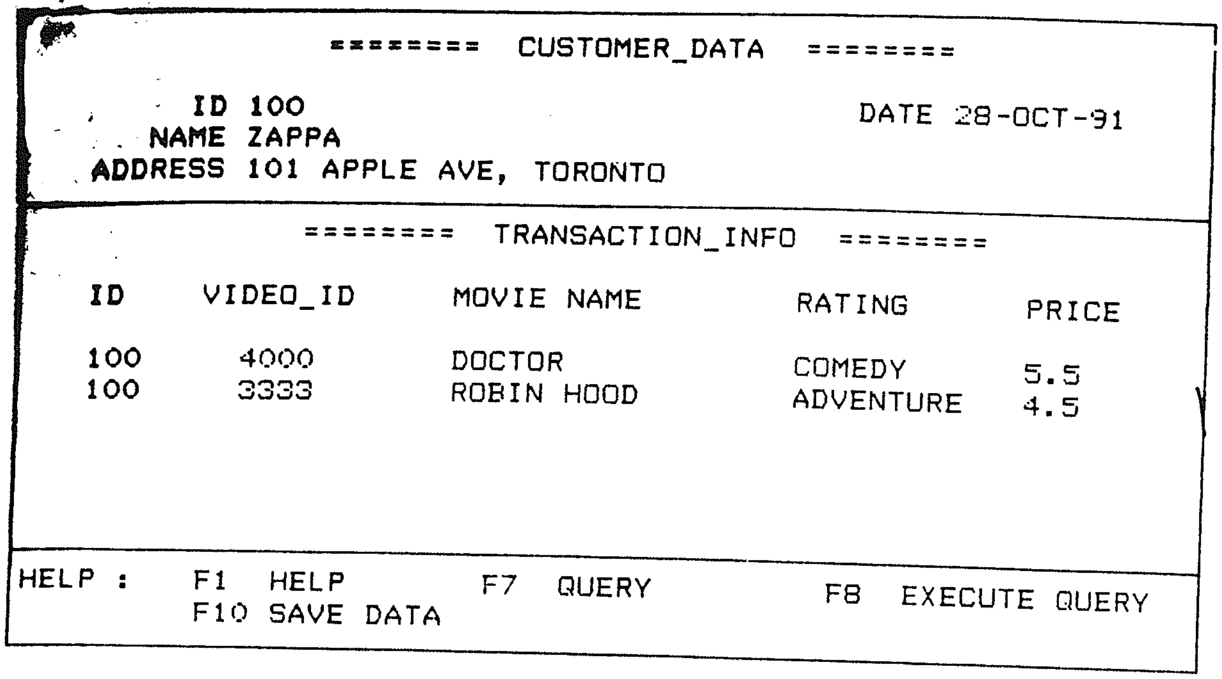
V.R.S. Case Study Detailed Specifications



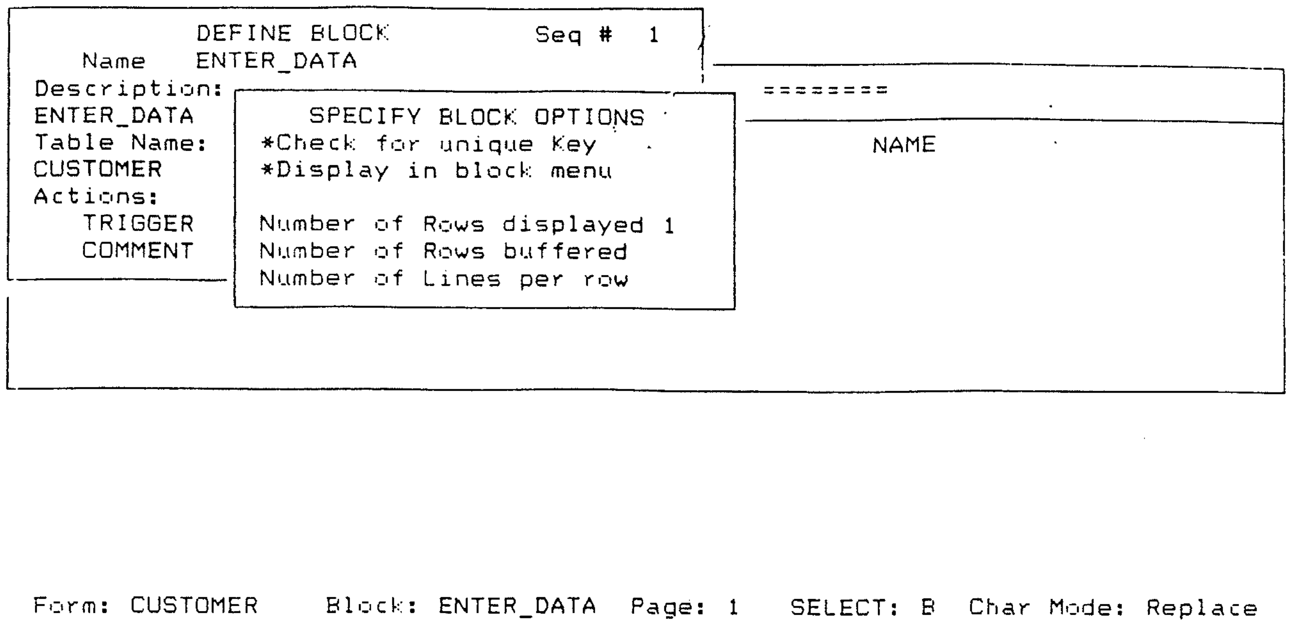
```
                    ========  ENTER_DATA  ========
          ID 100                                      NAME ZAPPA
  CREDIT_RATING GOOD
       ADDRESS 101 APPLE AVE, TORONTO
```



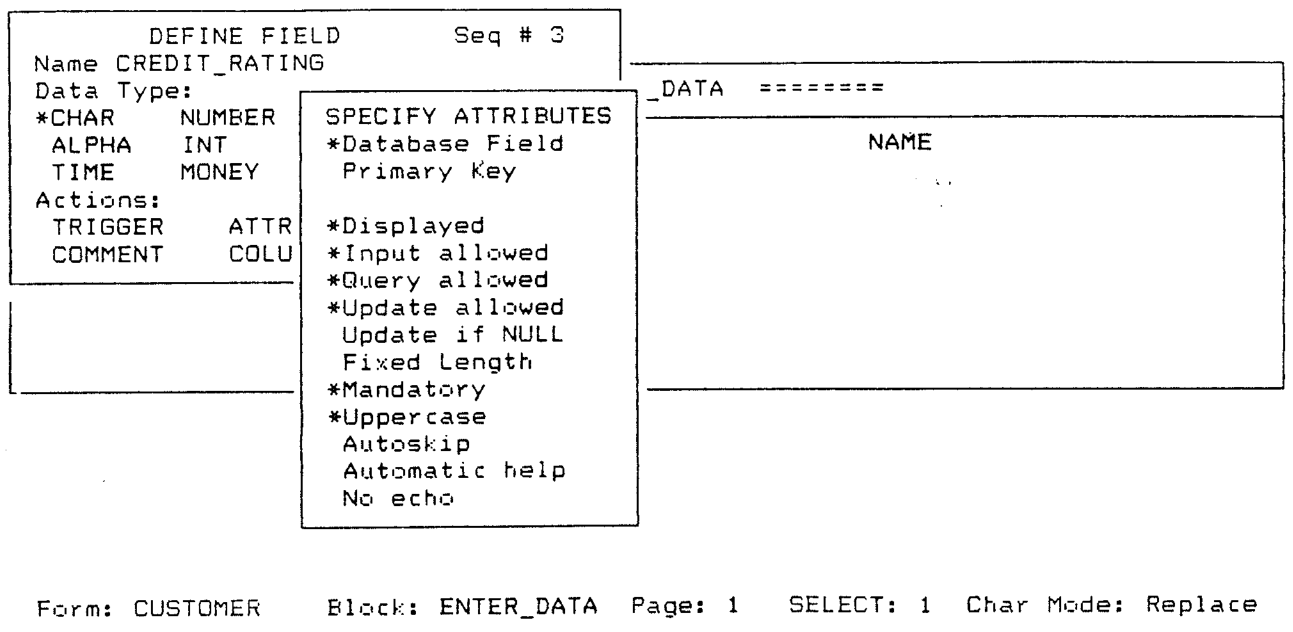
```
                    ========  ENTER_DATA  ========
     VIDEO_ID 5000                          MOVIE_NAME INDIANA JONES
        TYPE ADVENTURE                         RENTAL 4.00
```



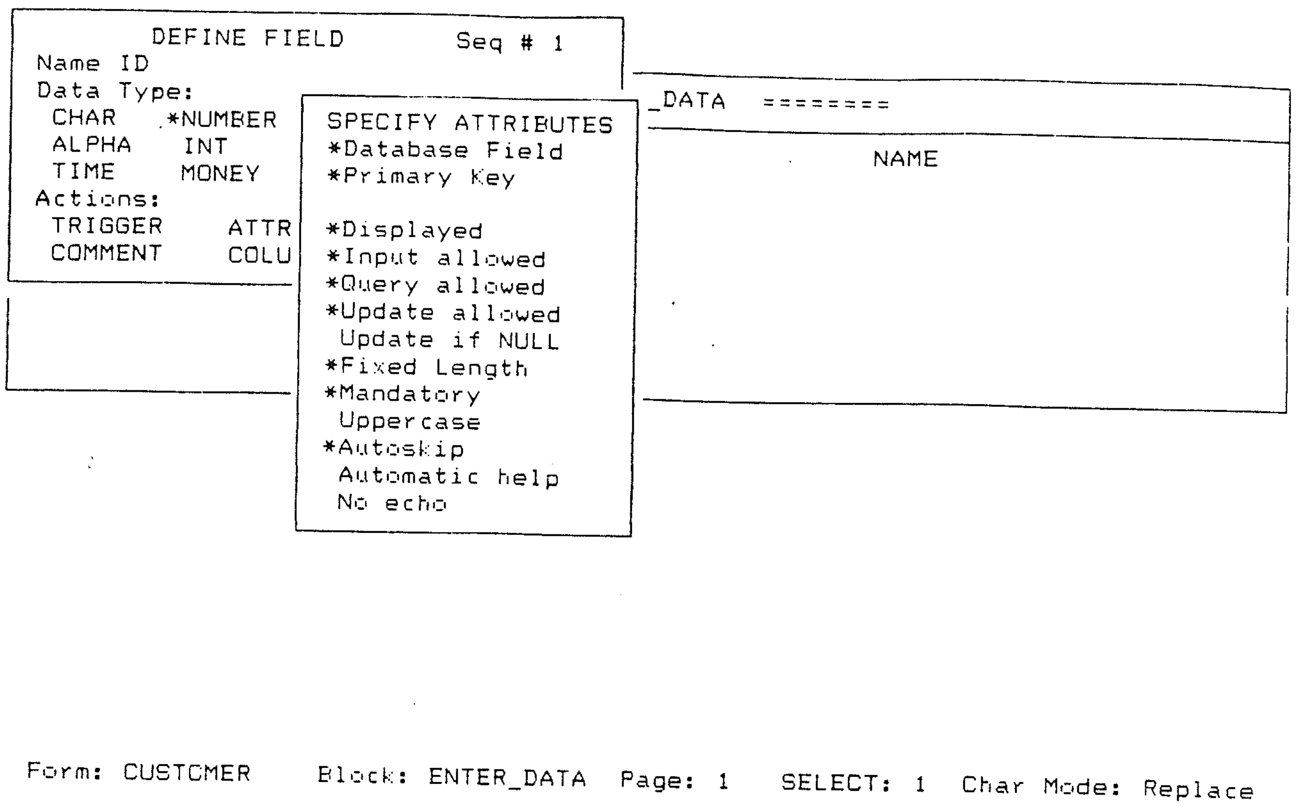
```
                ========  CUSTOMER_DATA  ========
          ID 100                                   DATE 28-OCT-91
        NAME ZAPPA
     ADDRESS 101 APPLE AVE, TORONTO
               ========  TRANSACTION_INFO  ========
   ID      VIDEO_ID      MOVIE NAME            RATING        PRICE

   100       4000        DOCTOR                COMEDY         5.5
   100       3333        ROBIN HOOD            ADVENTURE      4.5

HELP :     F1  HELP         F7   QUERY             F8   EXECUTE QUERY
           F10 SAVE DATA
```



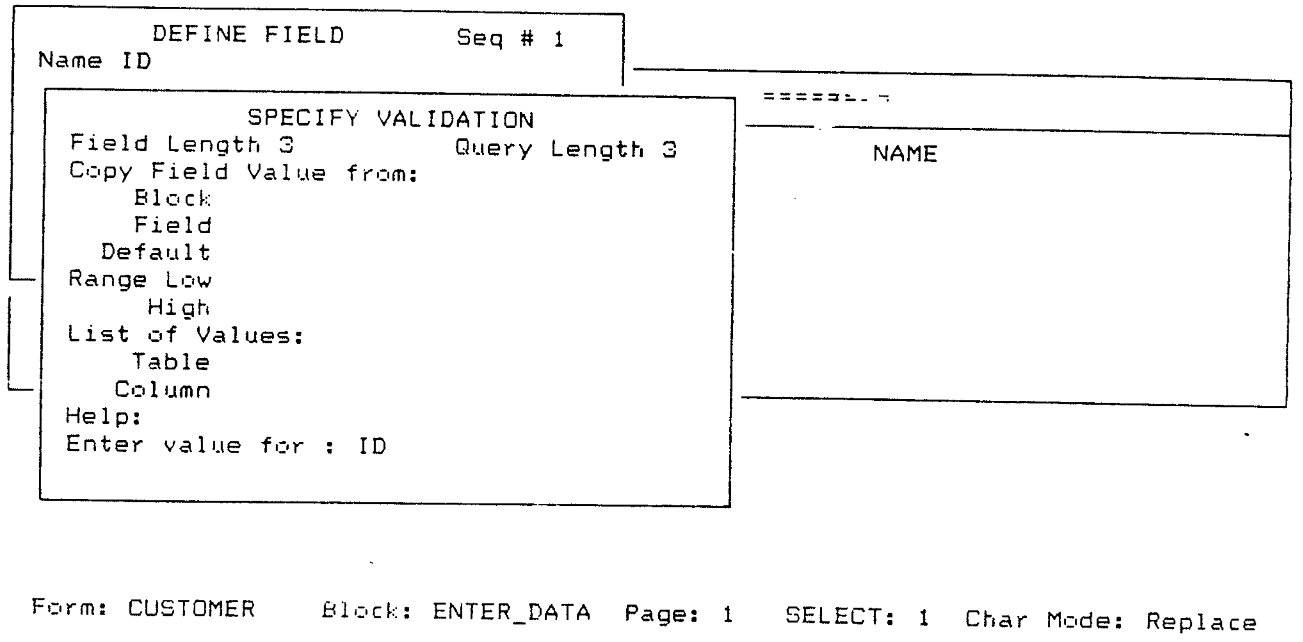
```
        DEFINE BLOCK          Seq #   1
   Name   ENTER_DATA
Description:                                     ========
ENTER_DATA   SPECIFY BLOCK OPTIONS
Table Name:  *Check for unique Key                    NAME
CUSTOMER     *Display in block menu
Actions:
   TRIGGER   Number of Rows displayed 1
   COMMENT   Number of Rows buffered
             Number of Lines per row

Form: CUSTOMER    Block: ENTER_DATA   Page: 1   SELECT: B   Char Mode: Replace
```



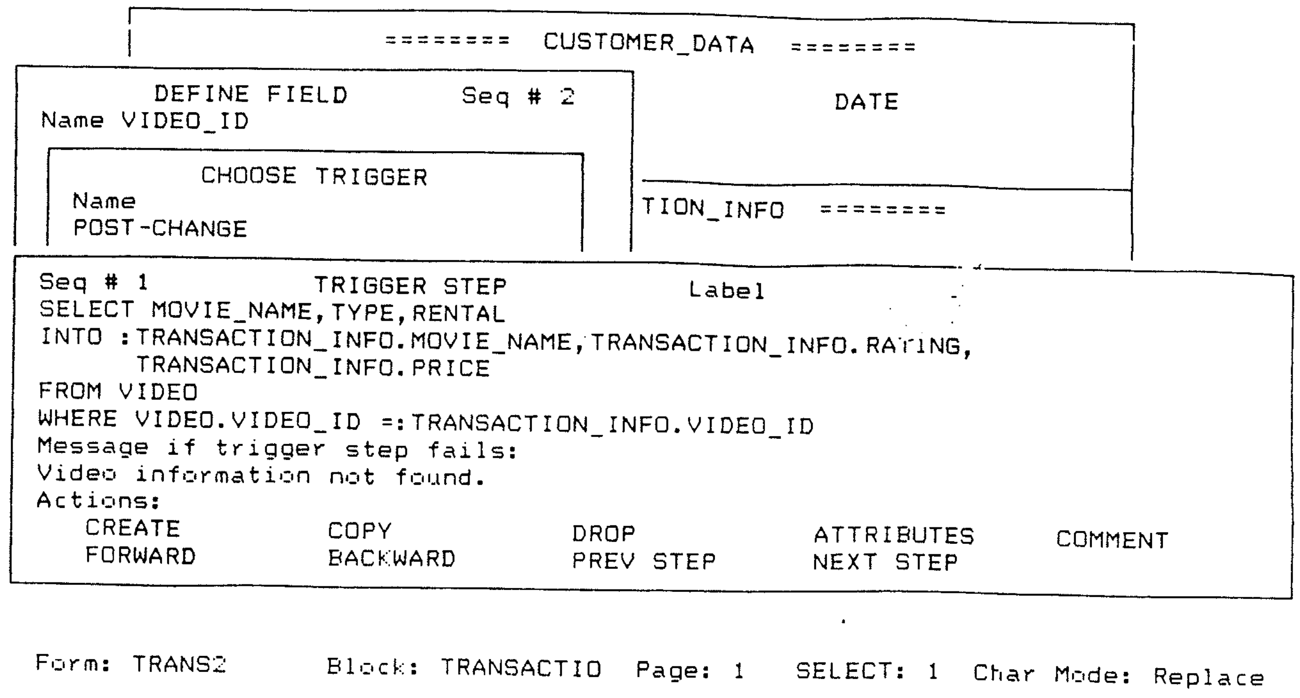
```
        DEFINE FIELD          Seq # 3
Name CREDIT_RATING
Data Type:                                   _DATA   ========
*CHAR     NUMBER   SPECIFY ATTRIBUTES
 ALPHA    INT      *Database Field                 NAME
 TIME     MONEY     Primary Key
Actions:
 TRIGGER     ATTR  *Displayed
 COMMENT     COLU  *Input allowed
                   *Query allowed
                   *Update allowed
                    Update if NULL
                    Fixed Length
                   *Mandatory
                   *Uppercase
                    Autoskip
                    Automatic help
                    No echo

Form: CUSTOMER    Block: ENTER_DATA   Page: 1   SELECT: 1   Char Mode: Replace
```



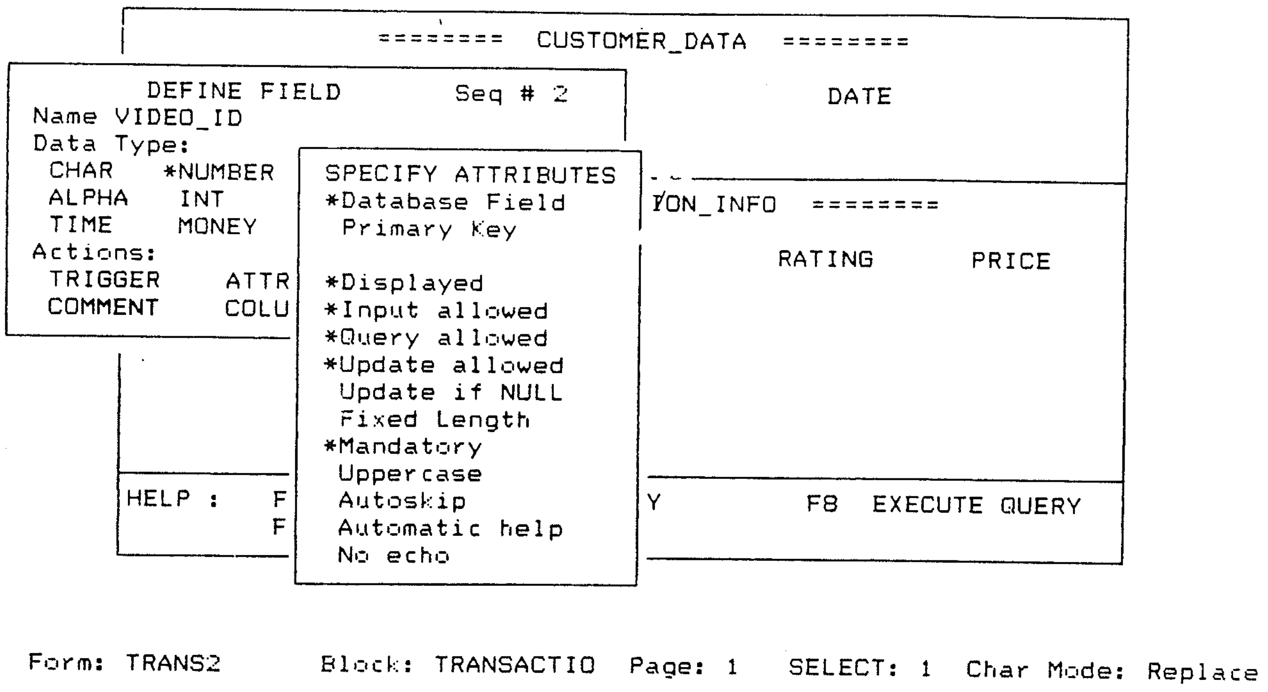
```
      DEFINE FIELD         Seq # 1
 Name ID
 Data Type:
  CHAR   *NUMBER    SPECIFY ATTRIBUTES      _DATA  ========
  ALPHA   INT       *Database Field
  TIME    MONEY     *Primary Key                    NAME
 Actions:
  TRIGGER    ATTR   *Displayed
  COMMENT    COLU   *Input allowed
                    *Query allowed
                    *Update allowed
                     Update if NULL
                    *Fixed Length
                    *Mandatory
                     Uppercase
                    *Autoskip
                     Automatic help
                     No echo

Form: CUSTOMER   Block: ENTER_DATA  Page: 1   SELECT: 1  Char Mode: Replace
```



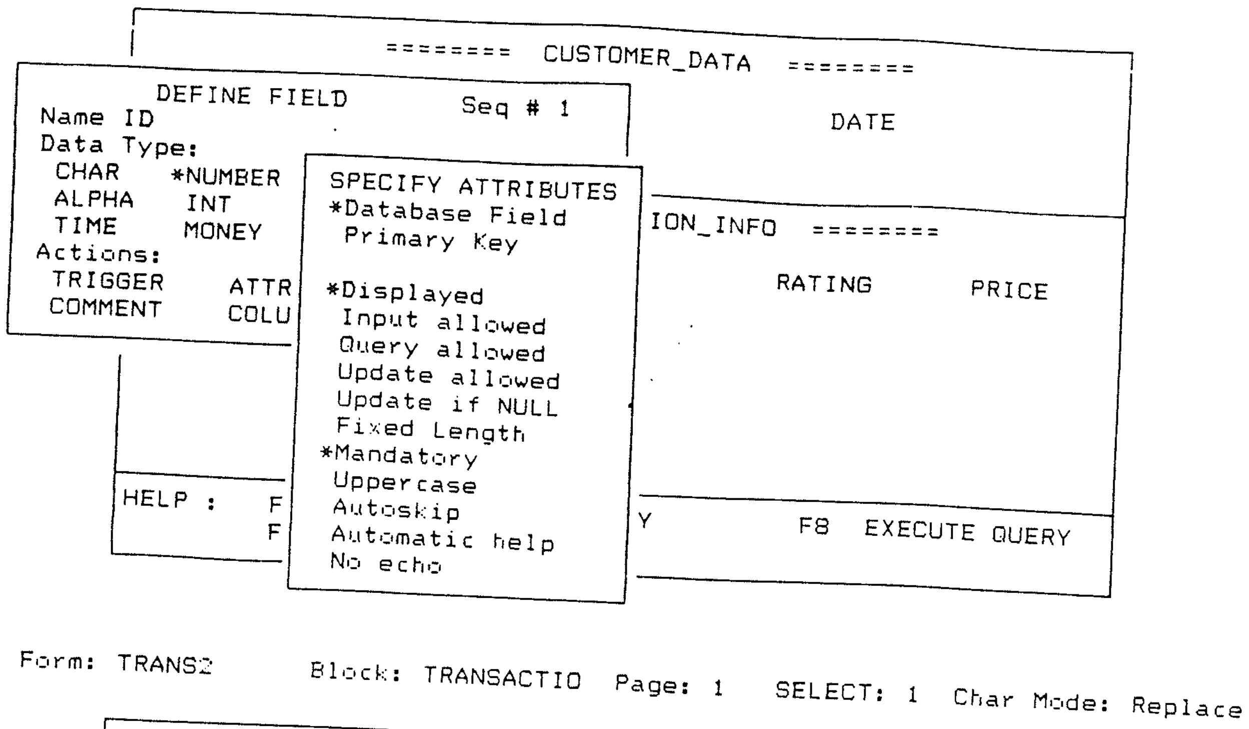
```
      DEFINE FIELD         Seq # 1
 Name ID
           SPECIFY VALIDATION                 ======-
  Field Length 3          Query Length 3
  Copy Field Value from:                      NAME
        Block
        Field
      Default
  Range Low
         High
  List of Values:
        Table
       Column
  Help:
  Enter value for : ID

Form: CUSTOMER   Block: ENTER_DATA  Page: 1   SELECT: 1  Char Mode: Replace
```



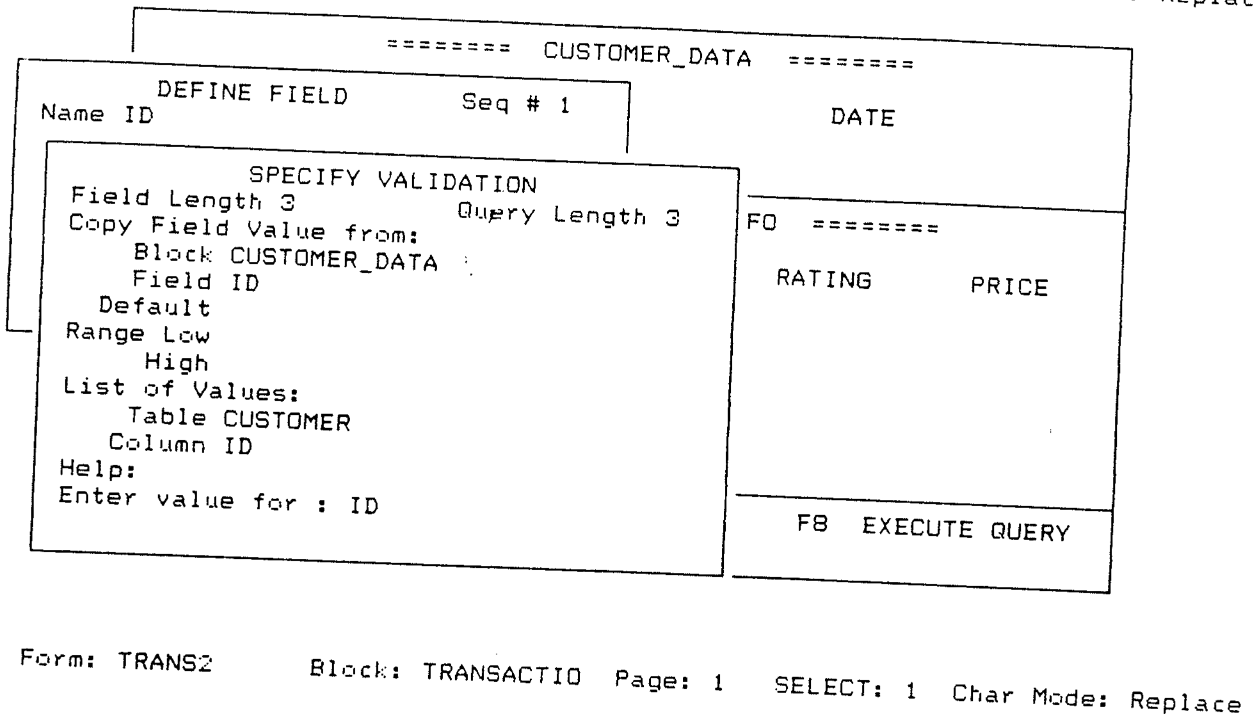
```
                         ========  CUSTOMER_DATA  ========
 DEFINE FIELD        Seq # 2
Name VIDEO_ID                                          DATE

        CHOOSE TRIGGER
  Name
  POST-CHANGE                            TION_INFO  ========

Seq # 1             TRIGGER STEP              Label
SELECT MOVIE_NAME,TYPE,RENTAL
INTO :TRANSACTION_INFO.MOVIE_NAME,TRANSACTION_INFO.RATING,
      TRANSACTION_INFO.PRICE
FROM VIDEO
WHERE VIDEO.VIDEO_ID =:TRANSACTION_INFO.VIDEO_ID
Message if trigger step fails:
Video information not found.
Actions:
   CREATE          COPY           DROP           ATTRIBUTES       COMMENT
   FORWARD         BACKWARD       PREV STEP      NEXT STEP

Form: TRANS2       Block: TRANSACTIO   Page: 1   SELECT: 1   Char Mode: Replace
```



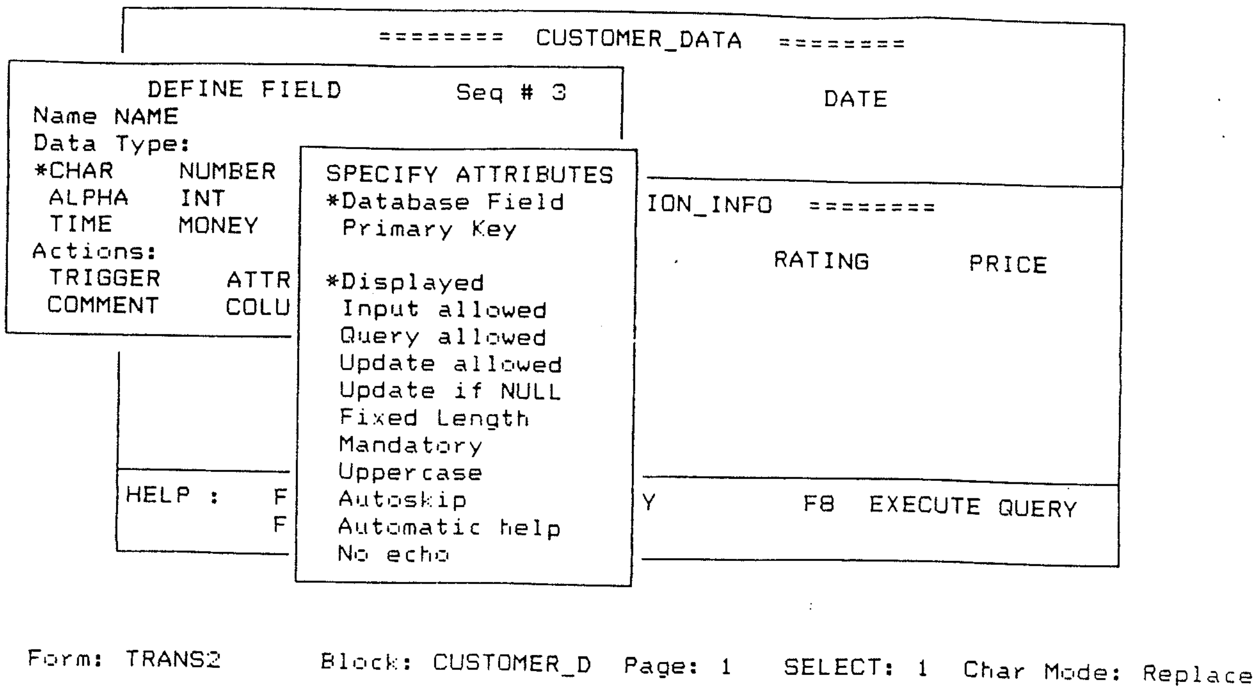
```
                         ========  CUSTOMER_DATA  ========
 DEFINE FIELD        Seq # 2
Name VIDEO_ID                                          DATE
Data Type:
 CHAR    *NUMBER   SPECIFY ATTRIBUTES
 ALPHA    INT      *Database Field       ION_INFO  ========
 TIME     MONEY     Primary Key
Actions:                                         RATING       PRICE
 TRIGGER     ATTR
 COMMENT     COLU  *Displayed
                   *Input allowed
                   *Query allowed
                   *Update allowed
                    Update if NULL
                    Fixed Length
                   *Mandatory
                    Uppercase
  HELP :     F      Autoskip          Y          F8  EXECUTE QUERY
             F      Automatic help
                    No echo

Form: TRANS2       Block: TRANSACTIO   Page: 1   SELECT: 1   Char Mode: Replace
```



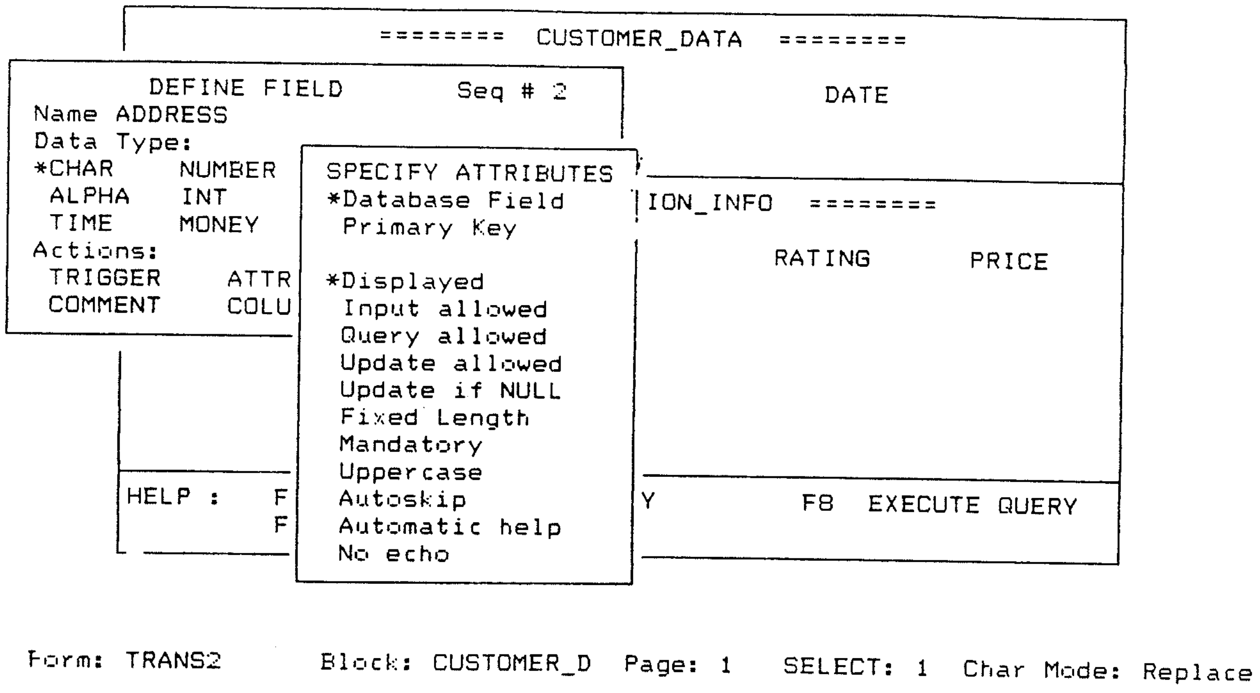
```
                    ========  CUSTOMER_DATA  ========
        DEFINE FIELD          Seq # 1
 Name  ID                                          DATE
 Data Type:
  CHAR     *NUMBER   SPECIFY ATTRIBUTES
  ALPHA     INT      *Database Field       ION_INFO  ========
  TIME      MONEY     Primary Key
 Actions:                                        RATING      PRICE
  TRIGGER      ATTR  *Displayed
  COMMENT      COLU   Input allowed
                      Query allowed
                      Update allowed
                      Update if NULL
                      Fixed Length
                     *Mandatory
 HELP :        F      Uppercase
               F      Autoskip
                      Automatic help   Y       F8   EXECUTE QUERY
                      No echo

Form: TRANS2     Block: TRANSACTIO  Page: 1   SELECT: 1  Char Mode: Replace
```



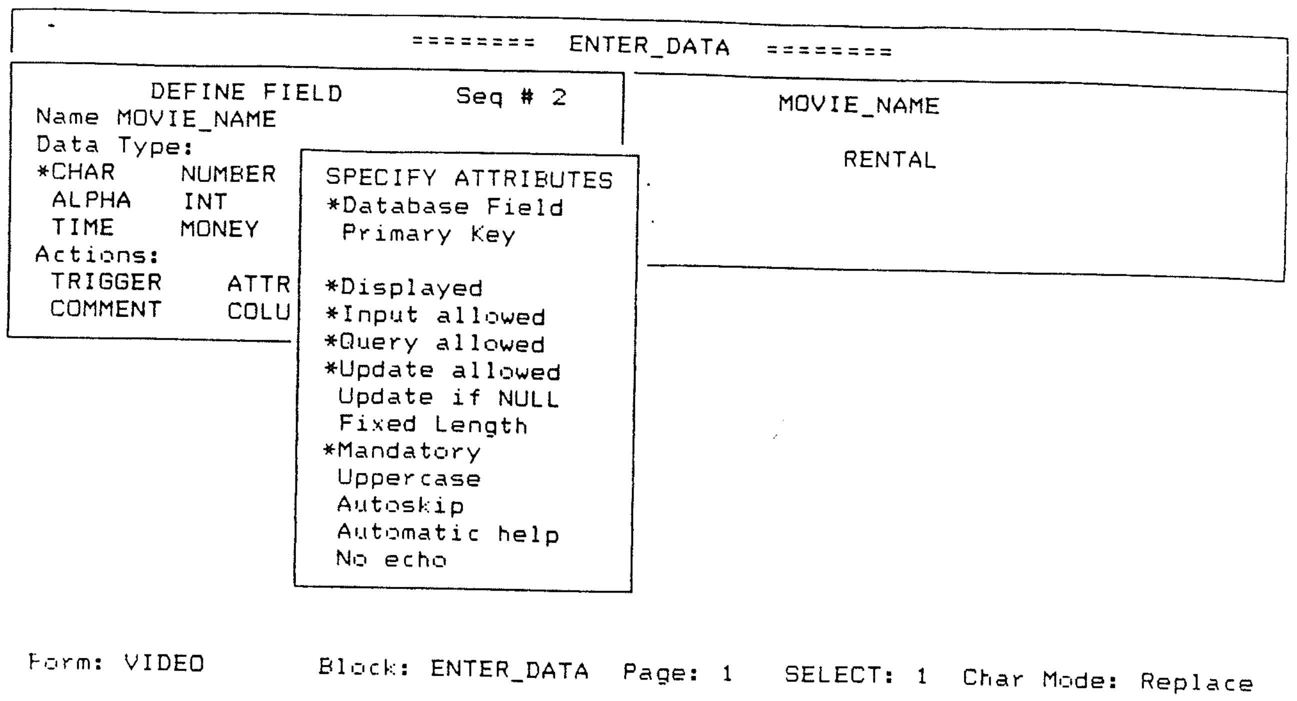
```
                    ========  CUSTOMER_DATA  ========
        DEFINE FIELD          Seq # 1
 Name  ID                                          DATE
               SPECIFY VALIDATION
 Field Length 3              Query Length 3   FO  ========
 Copy Field Value from:
      Block CUSTOMER_DATA                     RATING      PRICE
      Field ID
   Default
 Range Low
       High
 List of Values:
      Table CUSTOMER
    Column ID
 Help:
 Enter value for : ID
                                                F8  EXECUTE QUERY

Form: TRANS2     Block: TRANSACTIO  Page: 1   SELECT: 1  Char Mode: Replace
```



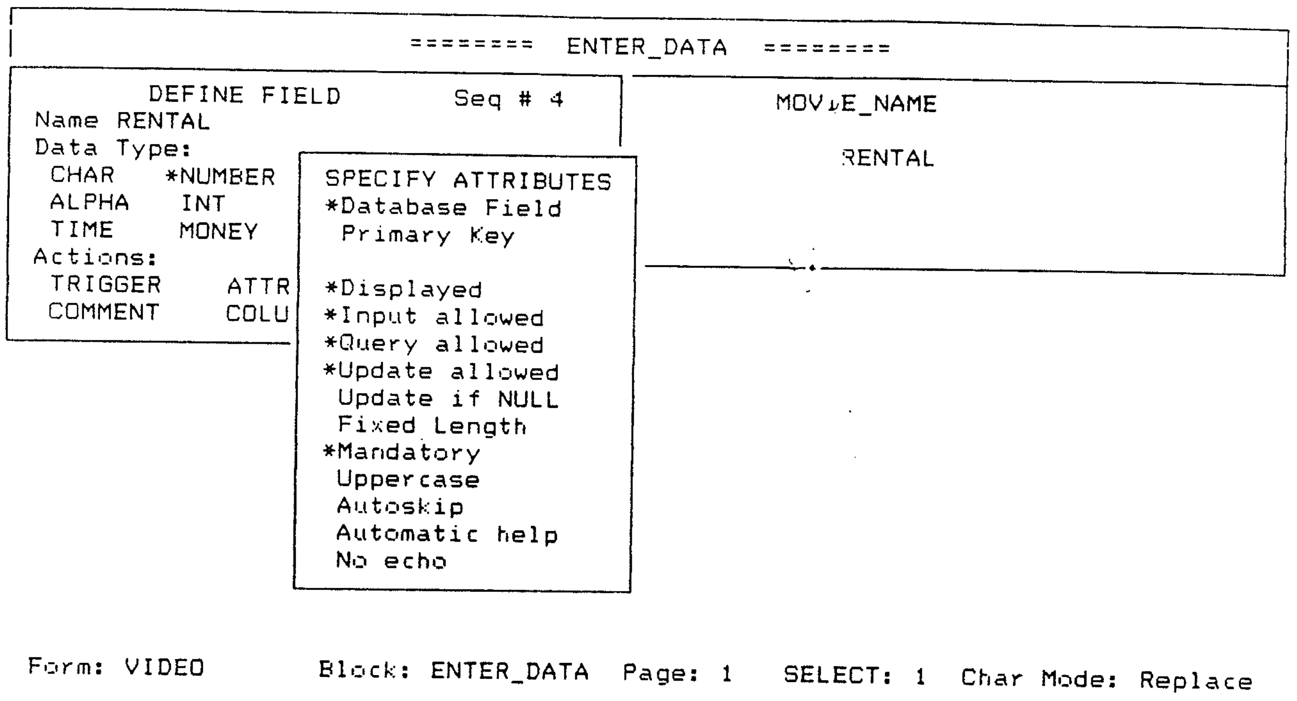
```
                    ========  CUSTOMER_DATA  ========
      DEFINE FIELD       Seq # 3                      DATE
 Name NAME
 Data Type:
 *CHAR     NUMBER  SPECIFY ATTRIBUTES
  ALPHA    INT     *Database Field    ION_INFO  ========
  TIME     MONEY    Primary Key
 Actions:                                       RATING      PRICE
  TRIGGER    ATTR  *Displayed
  COMMENT    COLU   Input allowed
                    Query allowed
                    Update allowed
                    Update if NULL
                    Fixed Length
                    Mandatory
                    Uppercase
 HELP :      F      Autoskip          Y        F8  EXECUTE QUERY
             F      Automatic help
                    No echo

Form: TRANS2     Block: CUSTOMER_D  Page: 1   SELECT: 1  Char Mode: Replace
```



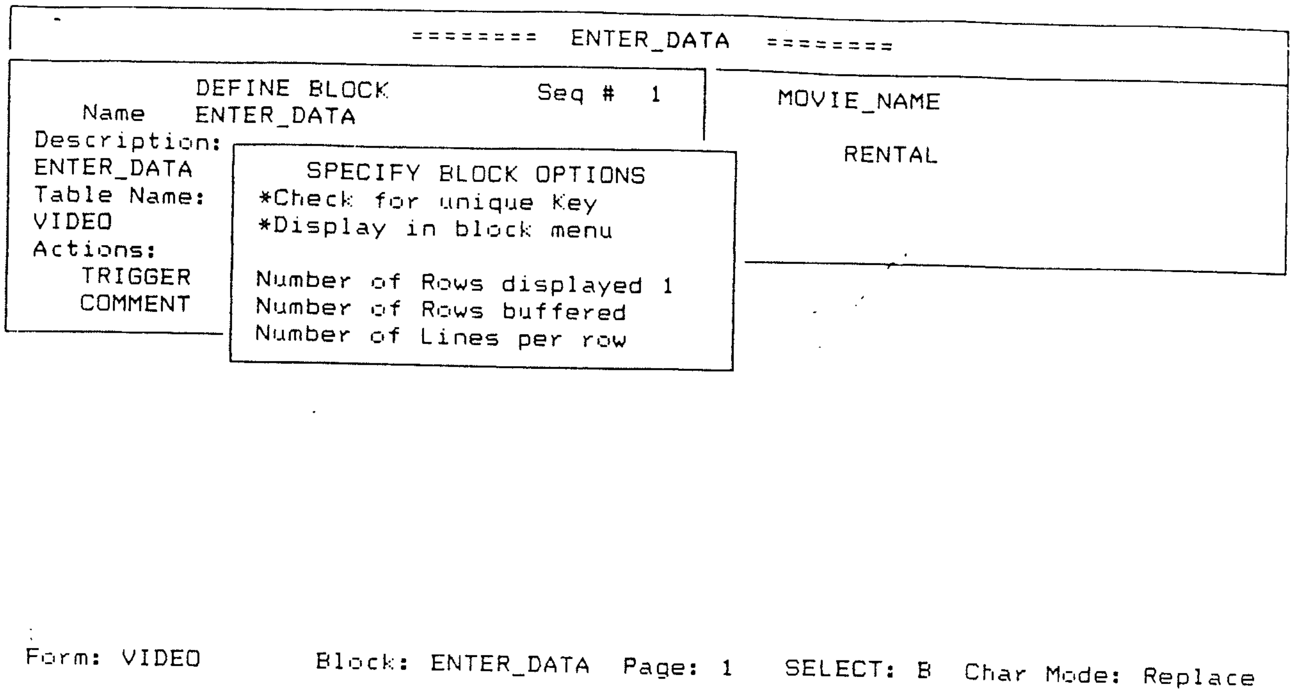
```
                    ========  CUSTOMER_DATA  ========
      DEFINE FIELD       Seq # 2                      DATE
 Name ADDRESS
 Data Type:
 *CHAR     NUMBER  SPECIFY ATTRIBUTES
  ALPHA    INT     *Database Field    ION_INFO  ========
  TIME     MONEY    Primary Key
 Actions:                                       RATING      PRICE
  TRIGGER    ATTR  *Displayed
  COMMENT    COLU   Input allowed
                    Query allowed
                    Update allowed
                    Update if NULL
                    Fixed Length
                    Mandatory
                    Uppercase
 HELP :      F      Autoskip          Y        F8  EXECUTE QUERY
             F      Automatic help
                    No echo

Form: TRANS2     Block: CUSTOMER_D  Page: 1   SELECT: 1  Char Mode: Replace
```



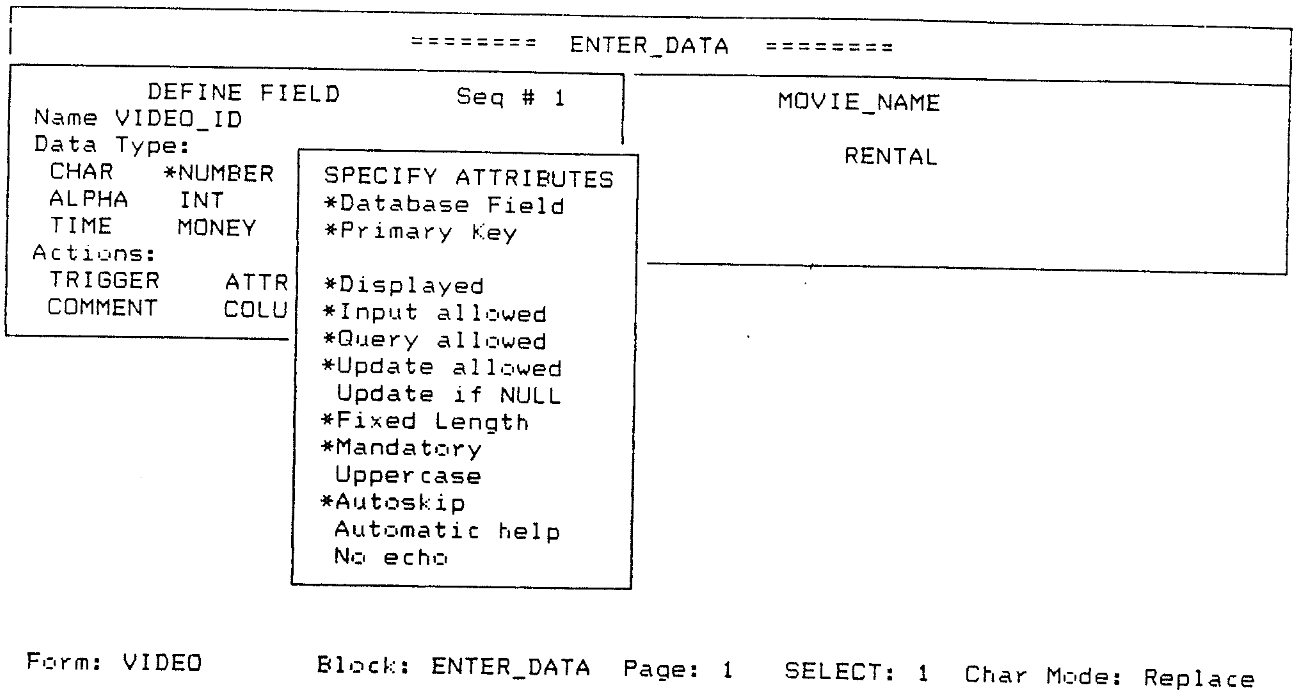
```
                         ========  ENTER_DATA  ========

        DEFINE FIELD         Seq # 2
 Name MOVIE_NAME                                    MOVIE_NAME
 Data Type:
 *CHAR     NUMBER   SPECIFY ATTRIBUTES                RENTAL
  ALPHA    INT      *Database Field
  TIME     MONEY     Primary Key
 Actions:
  TRIGGER     ATTR  *Displayed
  COMMENT     COLU  *Input allowed
                    *Query allowed
                    *Update allowed
                     Update if NULL
                     Fixed Length
                    *Mandatory
                     Uppercase
                     Autoskip
                     Automatic help
                     No echo

Form: VIDEO      Block: ENTER_DATA  Page: 1   SELECT: 1  Char Mode: Replace
```



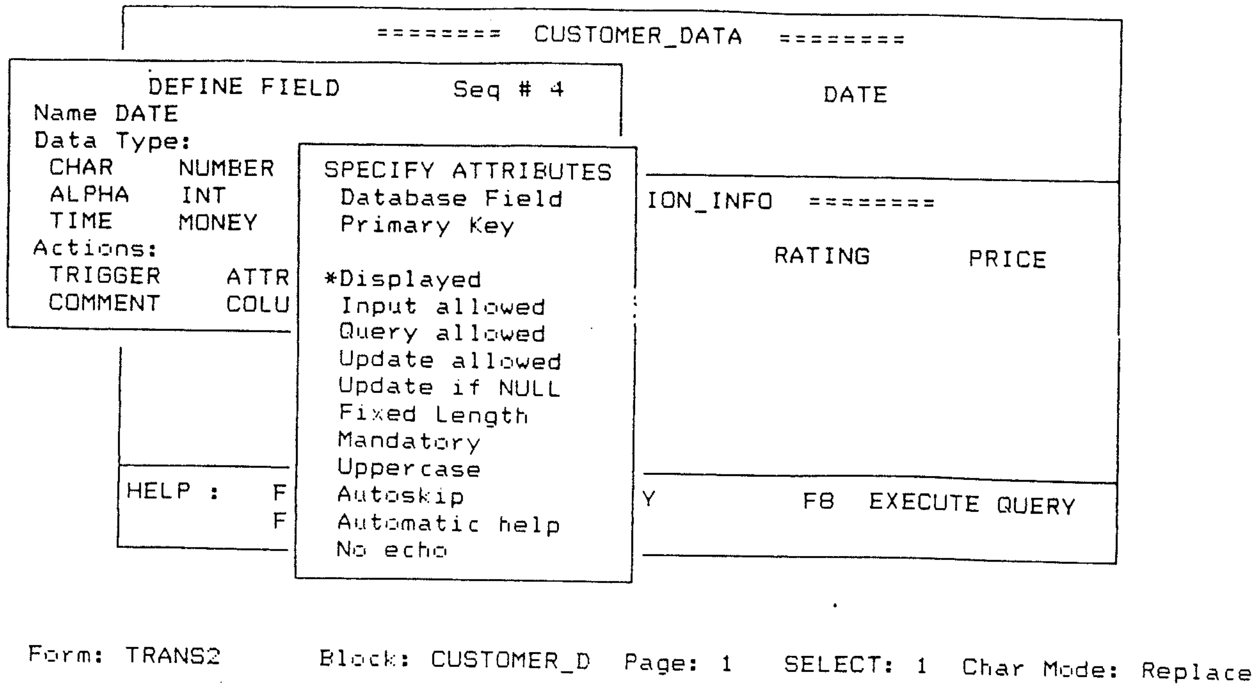
```
                         ========  ENTER_DATA  ========

        DEFINE FIELD         Seq # 4
 Name RENTAL                                        MOVIE_NAME
 Data Type:
  CHAR    *NUMBER   SPECIFY ATTRIBUTES                RENTAL
  ALPHA    INT      *Database Field
  TIME     MONEY     Primary Key
 Actions:
  TRIGGER     ATTR  *Displayed
  COMMENT     COLU  *Input allowed
                    *Query allowed
                    *Update allowed
                     Update if NULL
                     Fixed Length
                    *Mandatory
                     Uppercase
                     Autoskip
                     Automatic help
                     No echo

Form: VIDEO      Block: ENTER_DATA  Page: 1   SELECT: 1  Char Mode: Replace
```



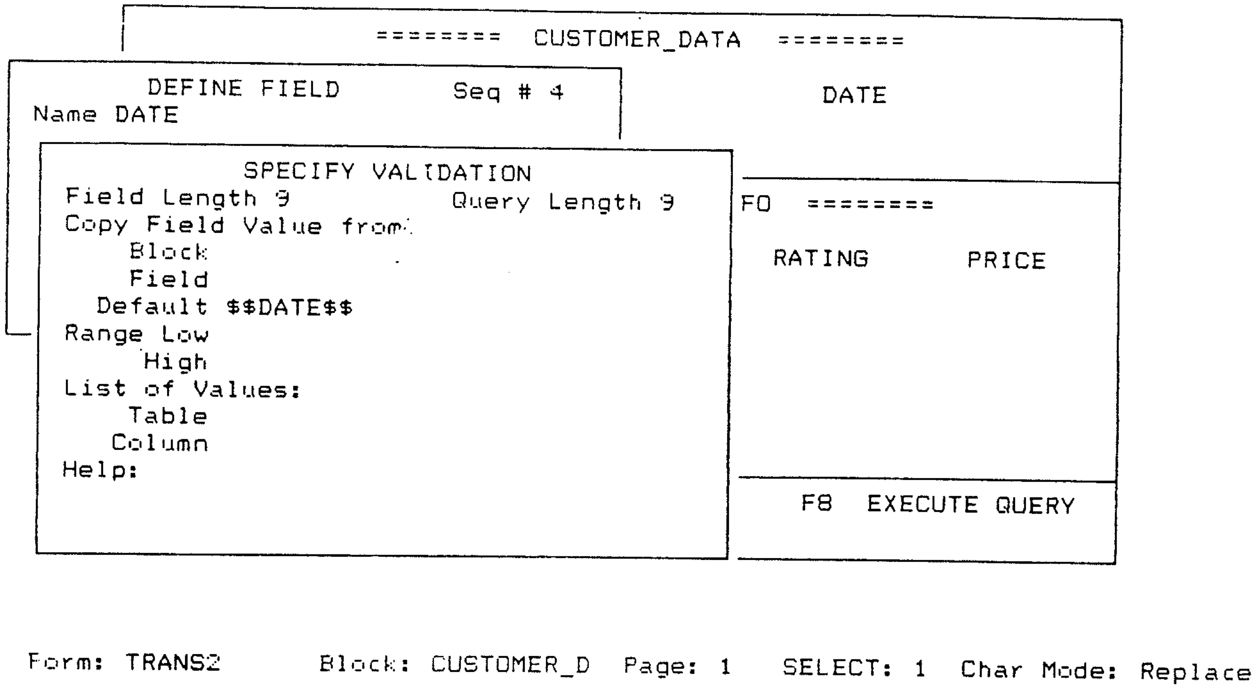
```
                         ========  ENTER_DATA  ========
         DEFINE BLOCK          Seq #  1        MOVIE_NAME
   Name   ENTER_DATA
Description:                                       RENTAL
ENTER_DATA        SPECIFY BLOCK OPTIONS
Table Name:    *Check for unique Key
VIDEO          *Display in block menu
Actions:
   TRIGGER     Number of Rows displayed 1
   COMMENT     Number of Rows buffered
               Number of Lines per row




Form: VIDEO     Block: ENTER_DATA  Page: 1   SELECT: B  Char Mode: Replace
```



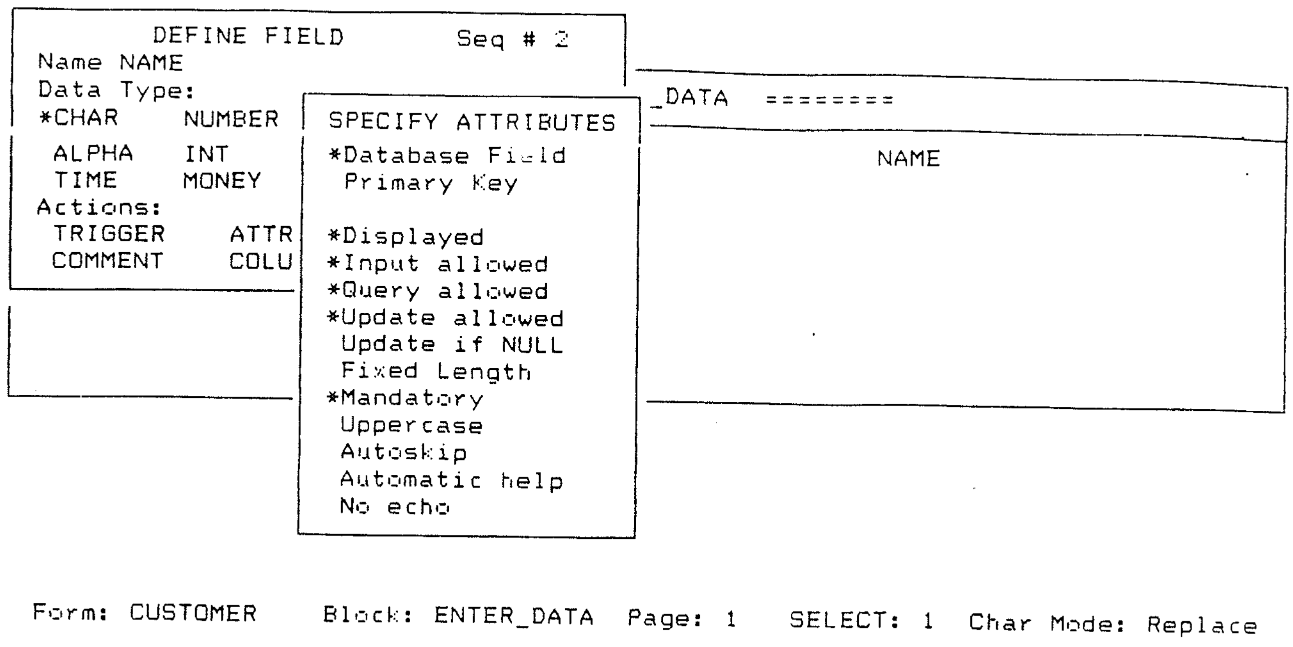
```
                         ========  ENTER_DATA  ========
        DEFINE FIELD        Seq # 1            MOVIE_NAME
Name VIDEO_ID
Data Type:                                         RENTAL
 CHAR    *NUMBER   SPECIFY ATTRIBUTES
 ALPHA    INT      *Database Field
 TIME     MONEY    *Primary Key
Actions:
 TRIGGER      ATTR *Displayed
 COMMENT      COLU *Input allowed
                   *Query allowed
                   *Update allowed
                    Update if NULL
                   *Fixed Length
                   *Mandatory
                    Uppercase
                   *Autoskip
                    Automatic help
                    No echo

Form: VIDEO     Block: ENTER_DATA  Page: 1   SELECT: 1  Char Mode: Replace
```



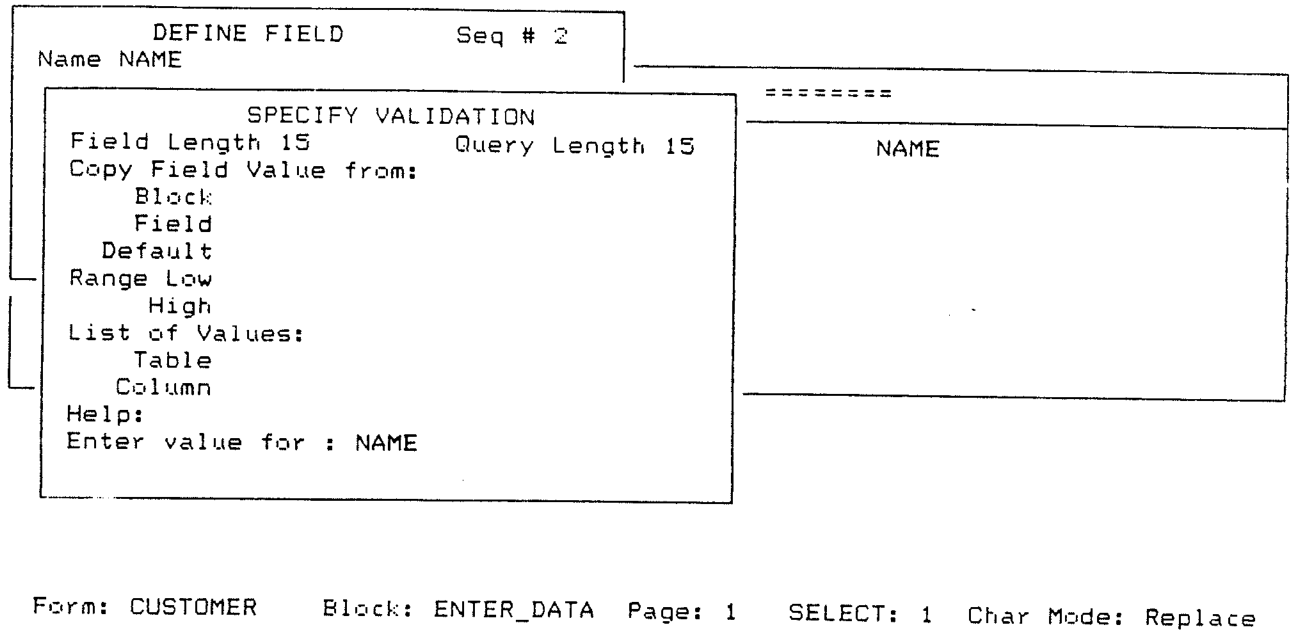
```
                    ========  CUSTOMER_DATA  ========
        DEFINE FIELD       Seq # 4                DATE
 Name DATE
 Data Type:              SPECIFY ATTRIBUTES
  CHAR     NUMBER         Database Field    ION_INFO  ========
  ALPHA    INT            Primary Key
  TIME     MONEY                                 RATING      PRICE
 Actions:                *Displayed
  TRIGGER      ATTR       Input allowed
  COMMENT      COLU       Query allowed
                          Update allowed
                          Update if NULL
                          Fixed Length
                          Mandatory
                          Uppercase
      HELP :       F      Autoskip          Y         F8   EXECUTE QUERY
                   F      Automatic help
                          No echo

Form: TRANS2      Block: CUSTOMER_D  Page: 1   SELECT: 1  Char Mode: Replace
```



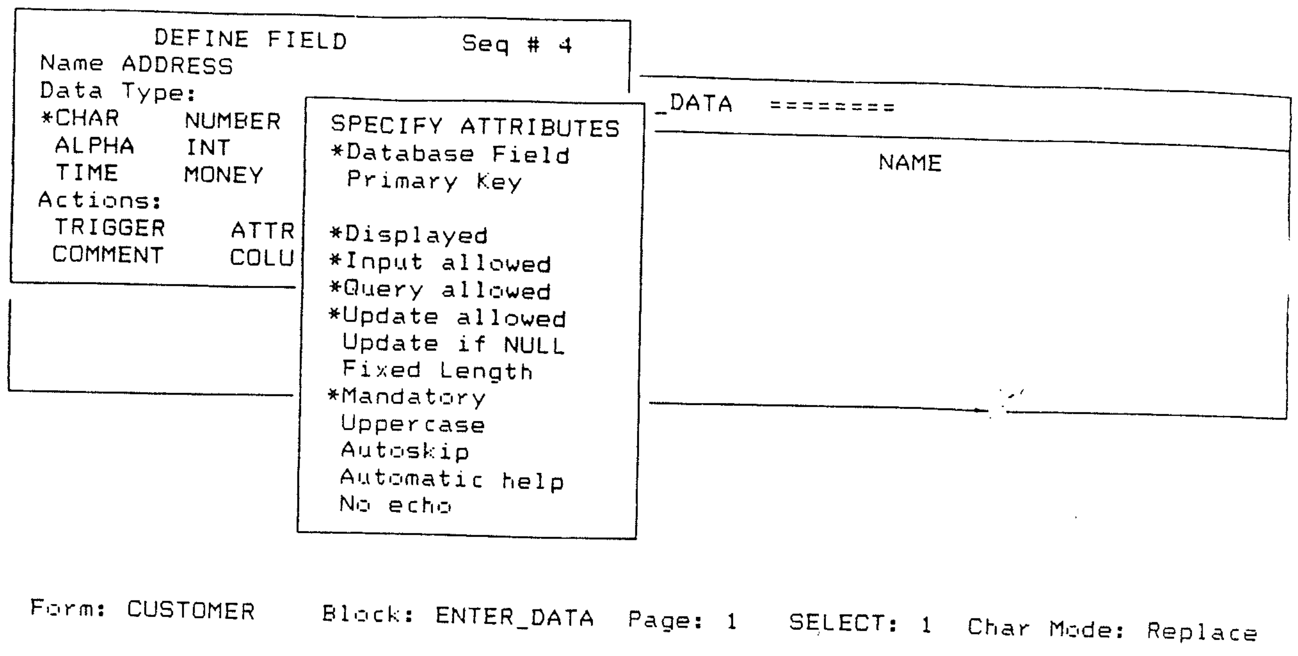
```
                    ========  CUSTOMER_DATA  ========
        DEFINE FIELD       Seq # 4                DATE
 Name DATE
              SPECIFY VALIDATION
  Field Length 9            Query Length 9     FO  ========
  Copy Field Value from:
      Block                                     RATING      PRICE
      Field
    Default $$DATE$$
  Range Low
       High
  List of Values:
      Table
     Column
  Help:
                                                  F8   EXECUTE QUERY

Form: TRANS2      Block: CUSTOMER_D  Page: 1   SELECT: 1  Char Mode: Replace
```



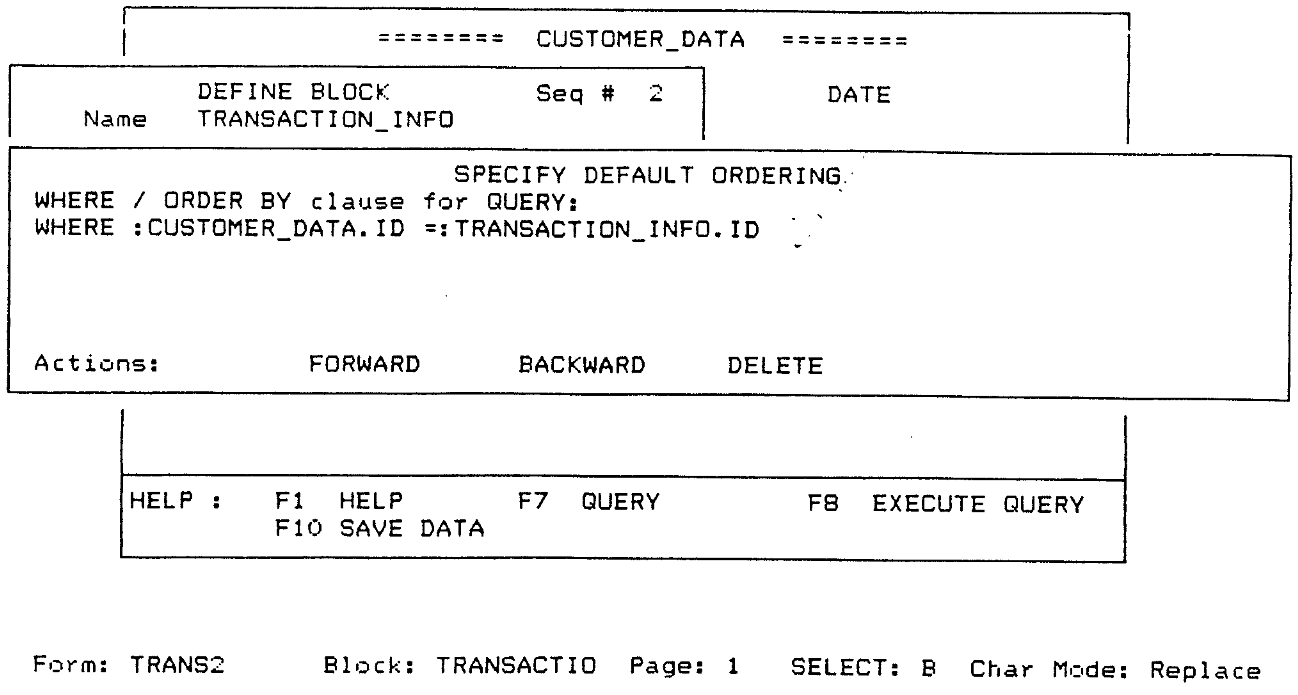
```
        DEFINE FIELD          Seq # 2
 Name NAME
 Data Type:
 *CHAR      NUMBER   SPECIFY ATTRIBUTES      _DATA  ========
  ALPHA     INT      *Database Field
  TIME      MONEY     Primary Key                     NAME
 Actions:
  TRIGGER      ATTR  *Displayed
  COMMENT      COLU  *Input allowed
                     *Query allowed
                     *Update allowed
                      Update if NULL
                      Fixed Length
                     *Mandatory
                      Uppercase
                      Autoskip
                      Automatic help
                      No echo

Form: CUSTOMER    Block: ENTER_DATA   Page: 1    SELECT: 1   Char Mode: Replace
```



```
        DEFINE FIELD          Seq # 2
 Name NAME
                                                  ========
             SPECIFY VALIDATION
  Field Length 15          Query Length 15           NAME
  Copy Field Value from:
       Block
       Field
     Default
  Range Low
        High
  List of Values:
       Table
      Column
  Help:
  Enter value for : NAME

Form: CUSTOMER    Block: ENTER_DATA   Page: 1    SELECT: 1   Char Mode: Replace
```



```
        DEFINE FIELD        Seq # 4
 Name ADDRESS
 Data Type:                                   _DATA   ========
 *CHAR     NUMBER   SPECIFY ATTRIBUTES
  ALPHA    INT      *Database Field                  NAME
  TIME     MONEY     Primary Key
 Actions:
  TRIGGER     ATTR  *Displayed
  COMMENT     COLU  *Input allowed
                    *Query allowed
                    *Update allowed
                     Update if NULL
                     Fixed Length
                    *Mandatory
                     Uppercase
                     Autoskip
                     Automatic help
                     No echo

Form: CUSTOMER    Block: ENTER_DATA   Page: 1    SELECT: 1   Char Mode: Replace
```



```
              ========  CUSTOMER_DATA  ========
         DEFINE BLOCK           Seq #  2              DATE
  Name   TRANSACTION_INFO
                          SPECIFY DEFAULT ORDERING
 WHERE / ORDER BY clause for QUERY:
 WHERE :CUSTOMER_DATA.ID =:TRANSACTION_INFO.ID



 Actions:           FORWARD       BACKWARD      DELETE

     HELP :    F1  HELP        F7  QUERY          F8  EXECUTE QUERY
               F10 SAVE DATA

Form: TRANS2      Block: TRANSACTIO  Page: 1    SELECT: B   Char Mode: Replace
```

```
SQL> GET CRCUSTOM
  1  CREATE TABLE CUSTOMER (
  2            ID           NUMBER(3)       NOT NULL,
  3            NAME         CHAR(15)        NOT NULL,
  4            CREDIT_RATING  CHAR(9)       NOT NULL,
  5*           ADDRESS      CHAR(30)        NOT NULL)
SQL> RUN
  1  CREATE TABLE CUSTOMER (
  2            ID           NUMBER(3)       NOT NULL,
  3            NAME         CHAR(15)        NOT NULL,
  4            CREDIT_RATING  CHAR(9)       NOT NULL,
  5*           ADDRESS      CHAR(30)        NOT NULL)

Table created.

SQL> GET CRVIDEO;
  1  CREATE TABLE VIDEO (
  2            VIDEO_ID         NUMBER(4)      NOT NULL,
  3            MOVIE_NAME       CHAR(15)       NOT NULL,
  4            TYPE             CHAR(9)        NOT NULL,
  5*           RENTAL           NUMBER(4,2)    NOT NULL)
SQL> RUN
  1  CREATE TABLE VIDEO (
  2            VIDEO_ID         NUMBER(4)      NOT NULL,
  3            MOVIE_NAME       CHAR(15)       NOT NULL,
  4            TYPE             CHAR(9)        NOT NULL,
  5*           RENTAL           NUMBER(4,2)    NOT NULL)

Table created.

SQL> GET CRTRANS;
  1  CREATE TABLE TRANS (
  2            ID                 NUMBER(3)      NOT NULL,
  3            VIDEO_ID           NUMBER(4)      NOT NULL,
  4*           DATE_RENTED        DATE           NOT NULL)
SQL> RUN
  1  CREATE TABLE TRANS (
  2            ID                 NUMBER(3)      NOT NULL,
  3            VIDEO_ID           NUMBER(4)      NOT NULL,
  4*           DATE_RENTED        DATE           NOT NULL)

Table created.
```

Appendix I

PFES Execution Trace

```
env_value
!  Testing ENV_16
!  !  env_rating
!  !  !  Testing ENV_1
!  !  !  !  ENVIRONMENT
!  !  !  !  !  (= PF_not_applicable CNF 100 )
!  !  !  Testing ENV_2
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_3
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_4
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_5
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_6
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_7
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_8
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_9
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_10
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_11
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_12
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_13
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_14
!  !  !  !  ENVIRONMENT
!  !  !  Testing ENV_15
!  !  !  !  ENVIRONMENT
!  !  !  (= average CNF 100 )
!  Testing ENV_17
!  Testing ENV_18
!  (= 1.00 CNF 100 )
env_value2
!  Testing ENV2_15
!  !  Environment2
!  !  !  (= PF_Not_Applicable CNF 100 )
!  (= 1.00 CNF 100 )
```

```
RELY
!  Testing C0
!  !  RELIABILITY_RATING
!  !  !  (= very_low CNF 100 )
!  (= 0.72 CNF 100 )
D_COMM
!  Testing COMM_0
!  !  DATA_COMMUNICATION
!  !  !  (= very_low CNF 100 )
!  (= 0.72 CNF 100 )
CHANGE
!  Testing F_C_0
!  !  FACILITATE
!  !  !  (= very_low CNF 100 )
!  (= 0.72 CNF 100 )
INTERFACE_COMPLX
!  Testing I_CMPLX_0
!  !  INTERFACE_COMPLEXITY
!  !  !  (= very_low CNF 100 )
!  (= 0.72 CNF 100 )
OPER_EASE
!  Testing O_EASE_0
!  !  OPERATIONAL_EASE
!  !  !  (= very_low CNF 100 )
!  (= 0.72 CNF 100 )
METHOD_VALUE
!  Testing MTHD_3A
!  !  TECHNIQUE
!  !  !  (= PF_has_no_influence CNF 100 )
!  (= 1.0 CNF 100 )
GT_TOOL_EXP_VALUE
!  Testing 8
!  !  GT_TOOL_EXP_IMPACT
!  !  GT_TOOL_EXP_RATING
!  !  !  (= VERY_HIGH CNF 100 )
!  Testing 9
!  !  GT_TOOL_EXP_IMPACT
!  !  GT_TOOL_EXP_RATING
!  Testing 10
!  !  GT_TOOL_EXP_IMPACT
!  !  GT_TOOL_EXP_RATING
!  Testing 11
!  !  GT_TOOL_EXP_IMPACT
!  !  GT_TOOL_EXP_RATING
!  Testing 12
!  !  GT_TOOL_EXP_IMPACT
!  !  GT_TOOL_EXP_RATING
!  (= 0.72 CNF 100 )
NOVELTY_VALUE
!  Testing 13
!  !  NOVELTY_IMPACT
!  !  NOVELTY_RATING
!  !  !  (= VERY_HIGH CNF 100 )
!  Testing 14
!  !  NOVELTY_IMPACT
!  !  NOVELTY_RATING
!  Testing 15
!  !  NOVELTY_IMPACT
!  !  NOVELTY_RATING
!  Testing 16
!  !  NOVELTY_IMPACT
!  !  NOVELTY_RATING
!  Testing 17
!  !  NOVELTY_IMPACT
!  !  NOVELTY_RATING
!  (= 0.72 CNF 100 )
TECHNIQUE_EFFECT
!  Testing MTHD_0
!  !  TECHNIQUE
!  Testing MTHD_1
!  !  TECHNIQUE
!  Testing MTHD_2
!  !  TECHNIQUE
PRACTICE_EFFECT
!  Testing MTHD_4
!  !  PRACTICE
!  !  !  (= PF_has_no_influence CNF 100 )
!  Testing MTHD_5
!  !  PRACTICE
!  Testing MTHD_6
!  !  PRACTICE
IMPACT
!  Testing MTHD_8
!  Testing MTHD_9
!  Testing MTHD_10
PERSONNEL_PF
!  Testing 1
!  !  DEVELOPER
!  !  !  (= DP_STAFF CNF 100 )
!  !  OVERALL_EXPVALUE
!  !  !  Testing 3
!  !  !  !  OV_EXP_IMPACT
! ! ! ! OVERALL_EXPRATING
! ! ! ! ! (= VERY_HIGH CNF 100 )
! ! ! Testing 4
! ! ! ! OV_EXP_IMPACT
! ! ! ! OVERALL_EXPRATING
! ! ! Testing 5
! ! ! ! OV_EXP_IMPACT
! ! ! ! OVERALL_EXPRATING
! ! ! Testing 6
! ! ! ! OV_EXP_IMPACT
! ! ! ! OVERALL_EXPRATING
! ! ! Testing 7
! ! ! ! OV_EXP_IMPACT
! ! ! ! OVERALL_EXPRATING
! ! ! (= 0.58 CNF 100 )
! (= (OVERALL_EXPVALUE*GT_TOOL_EXP_VALUE*NOVELTY_VALUE) CNF 100 )
! (= (RELY*D_COMM*CHANGE*INTERFACE_COMPLX*OPER_EASE) CNF 100 )
! (= (ENV_VALUE*ENV_VALUE2) CNF 100 )
! (= (PERSONNEL_PF*METHOD_VALUE*AF*EV) CNF 100 )
```