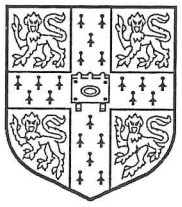
# Quantitative Functional Complexity Analysis of Commercial Software Systems

Stephen G. MacDonell

Clare College, Cambridge



A dissertation submitted to
the University of Cambridge
for the degree of
Doctor of Philosophy

Department of Engineering, University of Cambridge

October 1992

# Quantitative Functional Complexity Analysis of Commercial Software Systems

Stephen G. MacDonell

Clare College, Cambridge

## Abstract

This dissertation describes the development and validation of a software complexity analysis strategy based on the functional requirements of commercial systems that are to be developed using automated tools and techniques.

Software complexity has long been acknowledged as having a significant impact on software product quality, in terms of the number of post-delivery errors, and on the development process, in terms of personnel effort requirements. Such has been the extent of this impact that more than ninety distinct techniques for complexity assessment have been proposed. Changes in development technology, however, mean that many are now obsolete; furthermore, problems of late derivation, subjectivity, environment and personnel dependence and a lack of validation have impeded the widespread acceptance of most techniques.

The increasing use of automated assistance in software development, however, has provided an opportunity for significant advances to be made in commercial software complexity assessment. Computer-aided software engineering (CASE) tools and application generators are now mature enough to enable extensive system generation to be performed—direct transformation from specifications to final systems is therefore possible. This process significantly reduces the influence of personnel and environmental factors on development, thus enabling more objective assessment to be undertaken. A specification-based complexity analysis strategy has therefore been developed and validated in this study. The effectiveness of the approach in the discrimination and estimation of development effort and post-delivery error occurrence has been tested using data taken from sixteen commercial projects developed by ten different organisations. The results of the analysis confirm the assertion that functional complexity indicators are related to both system effort requirements and the likelihood of post-delivery system errors. It is therefore recommended that the results should be acted upon by project managers, and that the analysis scheme should be tested with data from other projects to further enhance the assistance that it provides.

# Preface

The work described in this dissertation was conducted at the Engineering Department and the Computer Laboratory of Cambridge University between October 1990 and October 1992. The analysis, results and discussions in this dissertation are the work of the author and include nothing which is the outcome of work done in collaboration. No part of this dissertation has been previously submitted to this or any other university for any degree. This dissertation contains approximately 50 000 words.

Many people deserve a vote of thanks for their assistance and encouragement over the duration of this project. Special mention must go to my supervisor, Dr Steve Young, whose patience, endurance and willingness to let me work with a free hand enabled this study to reach a successful conclusion under somewhat trying circumstances. The crucial assistance provided by William Mackaness and Steve King during the development of this dissertation has also been greatly appreciated.

The involvement of commercial software development sites was clearly an essential component of this study. I would therefore like to express my gratitude to those organisations that were bold enough to allow me access to their development records, and more specifically to those individuals responsible—Dave Rourke; Kate Head, Ben Stevens and Richard Neville; Deborah Kingsbury; Bob Austin and Simon Shiu; Margaret Belson and Steve Devonald; Keir McClelland; Nigel Gale; Steve King; Pete Brady; Lesley Butler, Paul Brombley and Ian Ashley. This study could not have proceeded without their help.

This study would also have been impossible but for the generous financial assistance of a number of organisations. I would therefore like to acknowledge the contributions of the Cambridge Commonwealth Trust, the New Zealand Vice-Chancellors Committee, British Telecom plc, Clare College, Cambridge and the Cambridge University Engineering Department.

Finally I would like to thank my family and friends. The self-imposed desire to live up to the hopes and expectations of others is a powerful incentive. This dissertation is therefore dedicated to my parents and to Sue, for their unceasing encouragement and their constant belief in my ability.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| 3GL | Third generation language |
| 4GL | Fourth generation language |
| CAPO | Computer-aided process organization |
| CASE | Computer aided software engineering |
| CDM | Control and definition modularization |
| CRUD | Create/read/update/delete |
| DBMS | Data base management system |
| DEO | Data elements flowing out of a system |
| DFD | Data flow diagram |
| ER | Entity relationship |
| ERD | Entity relationship diagram |
| FDH | Functional decomposition hierarchy |
| FP | Functional primitive |
| FPA | Function point analysis |
| GQM | Goal/question/measure |
| IE | Information engineering |
| LMS | Least-median-squares |
| LOC | Lines of code |
| LS | Least(-mean)-squares |
| MGM | Metrics guided methodology |
| MIS | Management information system |
| RE | Inter-object relationships |
| RLS | Reweighted-least-squares |
| RSL | Requirements specification language |
| SARA | System architect apprentice |
| STES | Specification-transformation expert system |

# Chapter 1

# Introduction

## 1.1 Introduction

Although still without standard definition, software complexity may be considered, for the purposes of this study at least, to be a determinant of the difficulty encountered by personnel in the development and maintenance of software systems. As a characteristic of the software development process, complexity has long been acknowledged as having a significant impact on several important product attributes, including quality, reliability and maintainability (Curtis [59]; Shepperd [222]; Hall and Preiser [106]). Complexity has also been recognised as an influential factor concerning the effective management of development projects (Colligan and Nevill [52]). This degree of importance has led to the development of more than ninety techniques, or metrics, for the assessment of complexity (Munson and Khoshgoftaar [181]). However, the effective application of a large number of these methods has been impeded by several problems—many are only useful at a very late stage in the development process, some appear to be more dependent on the development methods used and on the individual style and ability of programmers than on the actual complexity of the software, and several are less than comprehensive in their assessment (Sorensen [228]; Magel [169]; Case [40]; Samson *et al.* [217]). Moreover, a large number of these techniques have been proposed with little industry-based empirical justification (Bush and Fenton [37]; Myers [183]), leading to widespread scepticism of metrics within the development industry (Kearney *et al.* [136]; Ince and Shepperd [125]).

The increasing use of automated software development environments in the commercial software domain, however, has reduced the influence of implementation methods and programmer abilities on development task difficulty (Tate and Verner [242]). It is therefore suggested that in an automated development environment complexity analysis may be performed solely on functional specification products rather than on the traditional products of the lower-level design and construction phases. The overall intention of this research, then, is the development and validation of a specification-based functional complexity analysis scheme applicable to interactive commercial systems. Since a specification may be viewed from a number of perspectives the analysis approach suggested here attempts to consider aspects

of each representation, thus making the assessment more comprehensive than in many previous methods. Validation will be based on data collected from several industry sources with experience of development automation, from both the public and private sectors. Thus the term 'commercial system' is considered here to include administrative and transaction-oriented systems from both the government and business domains. It is envisaged that the results obtained from a statistical examination of the analysis scheme data and the associated project management records will provide evidence of relationships between functional complexity levels and project development tasks.

The remainder of this chapter describes the background to this research in terms of software development approaches, in order to provide a basis for the subsequently proposed objectives. It also considers software complexity as a general concept and discusses the impact that complexity is thought to have on current development processes. General approaches to complexity measurement are then examined so that an appropriate foundation for new assessment methods can be determined. Chapter 2 describes specification techniques that are widely used in the determination of commercial system requirements. These techniques provide the representations upon which the proposed analysis scheme is based. Advances in software development automation, which have had a significant impact on development difficulty, are also investigated. The third chapter then examines in greater detail current complexity analysis methods that are based on functional requirement specifications as represented in the notations described in Chapter 2. In light of the problems identified in this discussion the new complexity analysis scheme is proposed in Chapter 4. Chapter 5 considers the theoretical validity of the proposed approach and introduces the procedures that are to be used in the empirical evaluation of the proposal. Chapter 6 presents the results and discusses the findings of the statistical examination. The final chapter then summarises the study and considers the overall conclusions that can be drawn from the results. Recommendations for future research are then made.

## 1.2 Software Development and Complexity

As prescriptive methods for the development of software have evolved over the brief history of computing, a number of techniques to make that development simpler, more efficient and more effective have been suggested. Initially, the focus of these suggestions was on improving system coding, as this was where the majority of defects were introduced and where resource use was greatest. Approaches following structured programming were therefore encouraged, to increase quality in the first instance, maintainability in the second, and productivity as a whole. As the focus shifted to one of software design improvement, new techniques emerged to assist this activity (Beane *et al.* [15]; Card *et al.* [39]). Assessments of module interaction and self-containment were suggested so as to provide an insight into the level of strength in software design structures (Myers [182]; Troy and Zweben [246]; Ince [124]). A high degree of strength, in terms of these often heuristic design attributes,

was perceived as having a positive impact on the ease with which a system could be constructed and maintained.

More recently, increasing emphasis has been placed on the requirements analysis phase of development. Getting the requirements correct is seen as one of the most important aspects of development, as the implementation of a wrong system, no matter how quickly, still produces a wrong system (Boehm *et al.* [26]; Harwood [116]; Bobbie [24]). Hence the system prototyping approach has emerged, enabling the rapid generation of mock-up systems that can be used to determine user-perceived problems at an earlier stage than before. The use of computer aided software engineering (CASE) techniques has further enhanced this effort, with several early phase, or front-end tools providing checking facilities for completeness and consistency in a specification. This capability, coupled with automatic code generation, has in many cases resulted in more rapid development of systems that provide functionality closer to that which is expected by the user (Williamson [261]; Rinaldi [203]).

As stated above, the progression from one development approach to the next was fuelled by the dual requirements for greater development productivity and for systems of higher quality. Software complexity was therefore soon recognised as an influential determinant of both productivity and quality. Thus as these development approaches evolved, various methods for measuring or assessing the complexity of software were also developed.

### 1.2.1 Software Complexity

Complexity, according to Chen [42], is the least known factor in programming; it is not easily measured and is often ignored in system planning. This is despite the fact that complexity is acknowledged as an essential aspect of all software (Brooks [32]). It is clearly possible to build systems with differing degrees of complexity to perform the same function. Every functional requirement, however, has an optimal solution that has an associated level of inherent complexity (Bersoff *et al.* [19]). It is this functional complexity that is the focus of this study. Firstly, however, it is important to examine software complexity as a general concept so that a sound basis for assessment can be developed.

Rather than provide a definition for software complexity, Waguespack and Badlani [256] (p 52) classify it as a discipline:

> Software complexity is an area of software engineering concerned with the identification, classification and measurement of features of software that effect the cost of developing and sustaining computer programs. As a human endeavor, programming is subject to behavioral and psychological factors that eventually lead to the study of the human thought processes.

The incorporation of psychology in this classification is important in distinguishing this area of interest from that of computational complexity. Psychological or conceptual complexity generally refers to the features of software that have an impact on the ease of development, use and understanding of software from the human perspective. Computational complexity, on the other hand, is concerned with the

quantitative assessment of problem solutions from a machine perspective, for example, algorithmic efficiency (Curtis *et al.* [61]; Curtis [59]; Ejiogu [75]). It should therefore be noted that computational complexity is outside the scope of the current study.

Due to its abstract and multi-dimensional nature (Bowman and Newman [30]), complexity has proved to be difficult to define in a precise and objective manner; in spite of this it has generally been accepted that software complexity is a major determinant of several other software product attributes, including quality and maintainability (Bishop and Lehman [21]; Curtis [59]; Yau and Collofello [267]). Intuitive relationships such as these have prompted the large number of investigations into various methods of complexity assessment. Complexity has also been recognised as having a significant effect on project management decisions, such as the allocation of implementation and testing resources, or in the assignment of priorities for system maintenance (Fetzer [83]; Munson and Khoshgoftaar [181]; Ivan *et al.* [126]). This is in response to the general expectation that a more complex piece of software will take longer to develop, will contain more errors and will be more difficult to maintain and enhance (Gremillion [103]; Henry and Lewis [119]; Brooks [32]). This understanding has long been acknowledged (Weissman [259], p 25):

> It was realized [Naur and Randell 1968, Buxton and Randell 1969] that complexity of programs must be drastically reduced to aid in their understanding and maintenance.

Ultimately, it is the overall development and maintenance costs that are affected by a product's complexity (DeMarco [68]; Paulson and Wand [196]). The cost model employed by Boehm and Papaccio [27] suggests that savings can be substantially increased if complexity is effectively controlled. This in turn can contribute to greater user satisfaction as well as to continued producer profitability (Bhide [20]). Figure 1.1 illustrates the overall influence that complexity is thought to impose on the software development and maintenance process. Investigations into software complexity assessment have therefore become more widespread in recent years as the software development community has come to acknowledge the degree of influence that complexity has on development tasks and on overall project management. Some of the more traditional methods of complexity assessment are therefore now addressed.

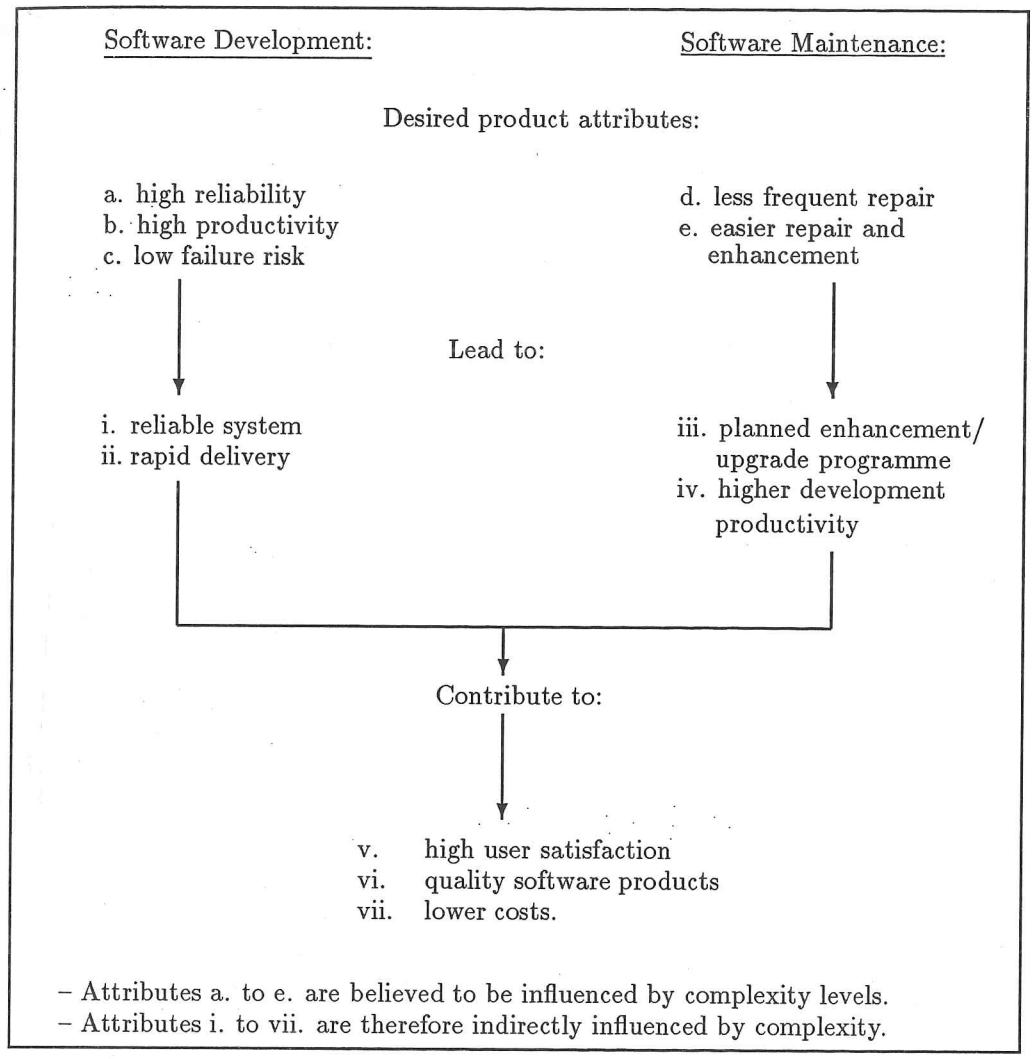



Figure 1.1: Complexity and software production

### 1.2.2 Traditional Approaches to Complexity Measurement

Over the last fifteen years, there has been an increasing amount of research into the use of quantitative software measurement in the assessment of development approaches. Software measurement generally involves the extraction of counts of various product and process attributes, based on the assumption that these counts may be useful in determining or estimating other development attributes. Thus the aim of many proposed complexity measurement techniques has been to provide product-based predictions of attributes such as development time or post-implementation error frequency.

The ultimate aim of any measurement procedure is to enable those responsible to effectively control aspects of their operating environment. Most previously proposed software complexity measures evaluate the degree to which a given feature exists within a system (Harrison [113]) with a view towards the prediction of, say, project effort or cost (Cherniavsky and Smith [47]). This approach is based on an

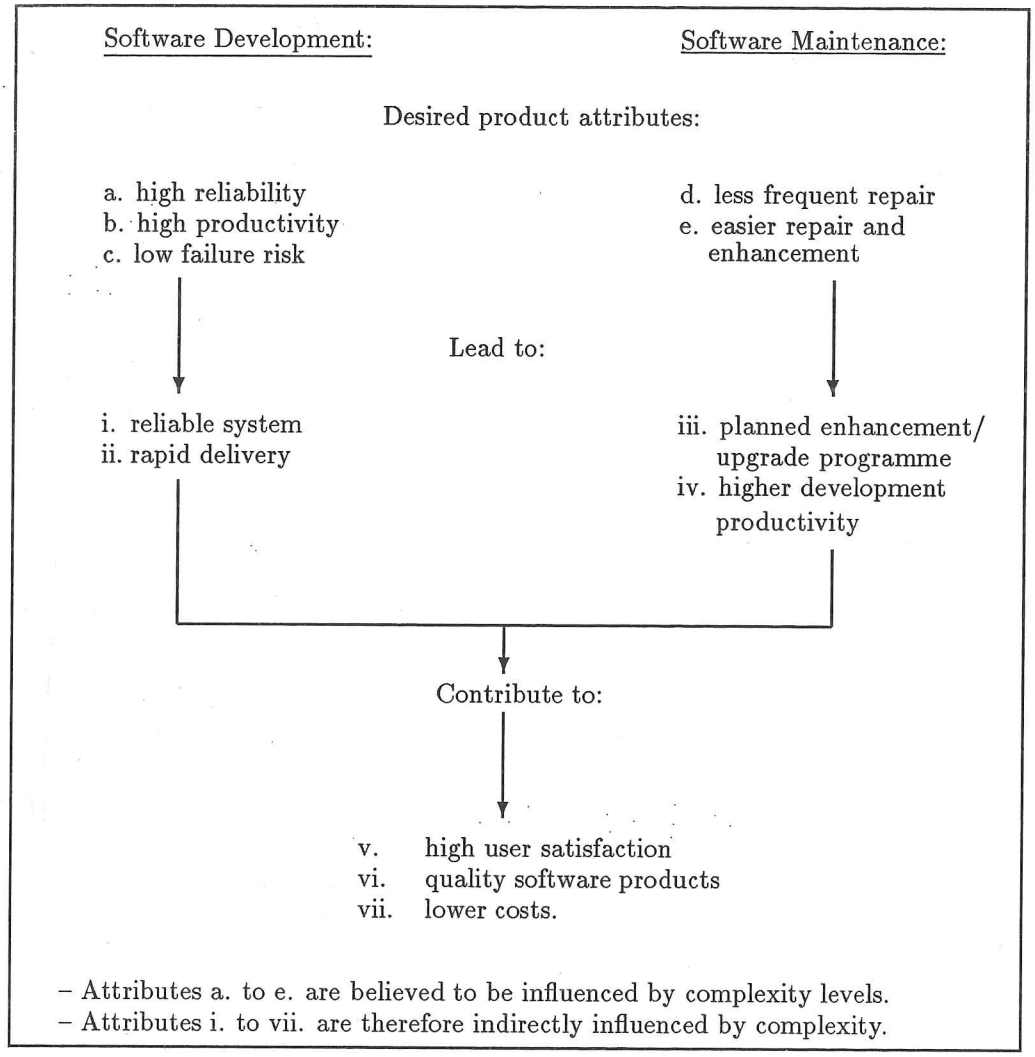



Figure 1.1: Complexity and software production

### 1.2.2 Traditional Approaches to Complexity Measurement

Over the last fifteen years, there has been an increasing amount of research into the use of quantitative software measurement in the assessment of development approaches. Software measurement generally involves the extraction of counts of various product and process attributes, based on the assumption that these counts may be useful in determining or estimating other development attributes. Thus the aim of many proposed complexity measurement techniques has been to provide product-based predictions of attributes such as development time or post-implementation error frequency.

The ultimate aim of any measurement procedure is to enable those responsible to effectively control aspects of their operating environment. Most previously proposed software complexity measures evaluate the degree to which a given feature exists within a system (Harrison [113]) with a view towards the prediction of, say, project effort or cost (Cherniavsky and Smith [47]). This approach is based on an

assumption that the frequency of the counted feature has some impact on the predicted attribute. This type of estimation is generally an ongoing, iterative process that is achieved in the following manner:

1. development of a predictive model based on existing data and/or intuitive considerations;
2. prediction of results based on the model;
3. collection of actual results;
4. comparison of actual and predicted results;
5. refinement of the model to accommodate the new results.

Although this type of procedure has traditionally been difficult to apply in the software development domain (Kitchenham and Walker [154]), the impact of increasing automation will reduce the influence of the many environmental, organisational and personnel factors that have caused extensive deviations in the past. More effective estimation should therefore be possible.

Unambiguous operational definitions of attributes such as complexity and quality remain obscure (Evangelist [76]; Bishop and Lehman [21]). Yet such has been the perceived importance of software complexity, coupled with changes in development methods, that well over ninety distinct techniques have so far been proposed for measuring this characteristic (Munson and Khoshgoftaar [181]). Generally each measure falls into one of the following five categories:

- lexical measures – derived through the counting of program code elements, for example, statements or operators
- topological measures – derived in terms of control-flow, data flow and nesting structure in programs
- structural measures – founded in the structural design of modular software to consider procedure interconnection and cohesion
- hybrid measures – measures that incorporate features from two or more of the previous classes
- functional measures – measures that may be derived from representations of a system's functional requirement.

Thus the earliest methods of complexity quantification were lexical, derived simply from the size of the programs being analysed. With the onset of structured programming, topological measures emerged, purporting to measure complexity in terms of control and data flow and nesting levels within the code. As the focus shifted to design, so the development of structural metrics began. Measures were generally based on the degree of module connection through control and data passing and hierarchical module calls. More recently, functional metrics have been proposed,

providing an indication of complexity based on some aspect of a system's functional representation. This is a traditional classification, in that measures are classed according to the way in which they are derived. More recently, however, it has become common to categorise measures based on the software product to which they are applied, or to combine measures of the same attribute together irrespective of the way in which the measures are derived. Factor analysis has also been performed on a set of more than thirty measures in an attempt to determine the essential dimensions of complexity (Munson and Khoshgoftaar [180]; [181]).

One of the most significant problems affecting accurate complexity measurement has been a lack of underlying theory relating to the understanding and programming processes. Evangelist [77] asserts that many proposed measures of complexity have been advocated and used with little theoretical foundation, most having no model on which to base their assumptions regarding human comprehension. This viewpoint appears to be widely supported—Davis [64], Kearney *et al.* [136] and Longworth *et al.* [165] all make similar observations. This situation makes the complete validation of techniques that are said to assess 'understandability' extremely unlikely.

A second obstacle to the more extensive acceptance of previously proposed complexity measures is the absence of metric validation over a wide range of languages and applications. Evaluation of most measures has been performed with software written in languages such as Fortran, Algol and PL/1 (Blaine and Kemmerer [23]; Han *et al.* [110]; Spratt and McQuilken [230]). Furthermore, despite the fact that many business applications are written in COBOL, and many scientific programs in C and C++, there has been comparatively little work performed on applications developed using these languages (Spratt and McQuilken [230]; Côté *et al.* [55]). This has resulted in a situation where metrics have been shown to be quite effective for a given language or type of application, but were then found to be completely inappropriate for a different problem area or implementation method (Rodriguez and Tsai [205]). General acceptance of these measures has therefore not occurred.

Many of the measures have also seen widespread criticism because of their single-aspect concentration. Although complexity appears to be a multi-faceted property that is influenced by a number of factors (Bowman and Newman [30]), most measures are based on only one aspect of a single software product (Magel [169]; Takahashi and Kamayachi [237]; Rodriguez and Tsai [206]). Sorensen [228] and Berns [18] suggest that this approach is too simplistic if objective and comprehensive indications are to be obtained. Yet Shen *et al.* [220] remark that the inclusion of a greater number of factors is unlikely to provide a better estimation method. The diverse combination of contributing factors has therefore impeded the development of broadly applicable operational measures (Vessey and Weber [256]). Jayaprakash *et al.* [131] suggest that researchers are still unsure as to how all the components of complexity can be assessed by one metric in a fair and balanced manner. So despite significant deficiencies in several well known metrics, many are still receiving extensive attention and consideration. Moreover, very little new work is being undertaken, in spite of the fact that many measures "...provide only a crude index of software complexity." (Kearney *et al.* [136], p 1050). Weyuker [260] stresses that the selection of appropriate metrics is made all the more difficult because it is not

always evident what a metric is actually measuring (also see Bollmann and Zuse [28] and Fenton and Kaposi [80] for further discussion of this issue). Often it is claimed that a measure may quantify characteristics such as implementation difficulty or the likelihood of maintenance, but these are themselves vague, non-operational features. The overall result is an absence of co-ordinated metric collection programmes in most software development organisations (Forte and Norman [89]).

All five measurement categories identified at the beginning of this section contain techniques that are undoubtedly related to at least some aspect of software complexity. The lexical metrics, such as those developed by Halstead [107], reflect the contribution of program size to the difficulty of software development and use. The topological measures are based on well-founded assumptions regarding the use of programs constructed under different procedural and data-flow strategies; see, for example, McCabe [175], Davis and LeBlanc [65] and Nejmeh [189]. Structural measures, such as those developed by Henry and Kafura [118] and Chapin [41], attempt to assess the effect of various design techniques on how difficult a system will be to develop and maintain. It is generally accepted that designs that employ a large degree of module connection and data passing are more difficult to implement and enhance than systems containing fewer connections. The hybrid measures, since they incorporate aspects of other measures, therefore incorporate the assumptions of the categories from which their components are derived. Metrics of this type are described by Harrison and Cook [114] and Li and Cheung [160]. Finally, the functional class of metrics is based on the assumption that a rigorously developed functional representation of a system will be directly related to the system that is finally implemented to provide that function (DeMarco [68]).

Within the commercial environment, however, it is unlikely that measures from all five categories will see continued use, particularly for project estimation tasks. Both the lexical and topological categories contain measures that are of little value in terms of early, useful feedback and subsequent estimation capabilities, due to their late stage of derivation. Furthermore, many suffer from counting method inconsistencies, they generally consider only one low-level aspect of overall complexity and they are inherently implementation dependent (Lennselius [158]; Bhide [20]). The majority of the hybrid class metrics are also affected by these drawbacks as many are derived from the lexical and topological classes. Extensive use of these methods in the new commercial software domain is therefore unlikely. This is not to say that measures from these classes will not be useful within other domains or when applied in a maintenance environment—topological metrics, for example, may be effective when applied to the formal specifications of scientific systems; or some of these measures could be useful in reverse engineering projects. This study, however, is only concerned with new commercial system development, and it is in this environment that these criticisms are said to apply.

The development and use of structural design metrics is a significant advancement on the previous techniques. The basis of these structural methods in generally accepted development principles, and the possibility of earlier determination, make the metrics from this class far more deserving of attention. Yet uptake of these methods by industry has been minimal, for a number of reasons—an absence of

automatic extraction capabilities has led to counting difficulties, as has the basis of techniques in varying design notations (Oman and Cook [194]). What is more, the apparent progress made towards having metrics that are derivable early in the development process and that are also implementation-independent has not been as significant as it at first appeared (Shepperd [222]). The utilisation of functional metrics is still in the formative stages, despite the widespread recognition of the need for early phase measurement. Subjectivity and environment dependence in particular are significant problems that have been associated with some of these methods including the technique developed by Albrecht [2], and the simplistic approach to complexity quantification that some have adopted also seems inadequate. This class of measures is discussed more fully in Chapter 3.

In spite of these problems, it is clear that the functional metric approach is the most promising for future complexity analysis. The increasing use of application generators and CASE tools now presents an opportunity for the development of analysis techniques that can overcome at least some of the problems associated with the other metric classes. Due to the degree of automation that these tools provide, the transformation from a system's functional requirement to an implementation is made much more straightforward (Symons [236]; Verner *et al.* [254]), thus reducing the impact of development personnel and implementation methods; the multi-dimensional nature of specifications enables the assessment of complexity from a number of perspectives, leading to a more comprehensive consideration; and requirement representations are among the first tangible products of the software development process, so measures taken from them are likely to be among the earliest available. Measures from functional representations have the potential to be useful in comparing the complexity of both complete systems and individual functions, and in assessing the impact that these levels of complexity have on the outcome of the development process. The increasing use of CASE tools in the development of commercial software should therefore enable automatic, implementation-independent assessment of functional complexity to be performed as an integral part of the software development and maintenance activities. This assertion may be more formally expressed in the following research objectives.

## 1.3 Research Objectives

Complexity influences both the software product, in terms of error-proneness, and the software development process, in terms of development effort. If relatively precise relationships can be established between early indicators of complexity and project management data, managers can then obtain a valuable insight into the likely outcome of a project in terms of effort expended and errors incurred. The overall aim of this study, then, is to develop and validate an appropriate specification-based functional complexity analysis scheme in order to determine relationships of interest to project managers. More specifically the study aims to achieve the following objectives:

- the determination of problems associated with previously proposed functional complexity assessment techniques
- the development of an analysis scheme that overcomes the failings of previous methods
- the determination of relationships between functional complexity indicators and project management data (relating to development effort and error occurrence)
- the early determination of relative functional complexity indicators (in terms of development effort and error occurrence) at both the system and individual function level
- the classification of systems and individual functions according to their likely project management consequences (in terms of development effort and error occurrence) based on functional complexity indicators
- the development of equations for the estimation of project management data (relating to development effort and error occurrence) based on functional complexity indicators.

The review of specification techniques in Chapter 2, along with the consideration of issues relating to functional analysis in Chapter 3, provides the basis for the analysis scheme as proposed in Chapter 4. This will represent the achievement of the first two objectives. The empirical validation of the proposed complexity analysis scheme, which appears in Chapter 6, will describe the achievement, in operational terms, of the remaining four objectives.

# Chapter 2

# Commercial Software Specification

## 2.1 Introduction

Software development in the commercial environment makes extensive use of modelling techniques, particularly in the early phases of the development process. At the specification stage this enables both users and developers to obtain abstract and concise representations of the requirements that a system is to fulfill. It is suggested here that representations such as these will be useful in providing relative indications of functional complexity, particularly within an automated development environment. This chapter therefore begins with an examination of literary support for the use of software specifications as the basis for the proposed analysis scheme. This is followed by a discussion of widely used specification techniques in commercial software development. The impact and role of development automation, in terms of computer aided software engineering (CASE) tools and fourth generation languages (4GLs), are also examined in relation to the specification techniques.

## 2.2 Support for Focus on Specifications

The conclusions made in the previous chapter regarding functional complexity assessment methods suggest the continued development of similar early-phase techniques. Shepperd [222] suggests that the current level of understanding relating to early phase complexity metrics is so limited that their use is almost exclusively restricted to a subjective form of quality assurance. With the increasing development of automatic analysis and design tools, however, this situation should begin to change (Côté *et al.* [55]) to reflect the growing need for pre-coding assessment (Lanphar [156]). It has long been acknowledged that indications of final product characteristics are needed well before the construction stage (Curtis [60]; Ottenstein [195]; Gaffney *et al.* [92]). The late availability and limited applicability of code-based metrics means that there is now little to support their use. This suggests that attention would be better directed towards design and specification measures (Ince

and Shepperd [125]; Fenton and Melton [81]).

The increased potential for feedback relating to the progress of development and to the error-proneness of emerging products is the main motivation behind the use of early-phase measures (Porter and Selby [197]; Compton and Withrow [53]). A large proportion of development effort is often spent on rework, due to functional difficulties that should be identified in the conceptual specifications (Brooks [32]) but only become apparent in the later development phases. As maintenance costs are far less significant in these early stages, there is clearly an economic requirement for the early detection of specification problems (Boehm and Papaccio [27]; Dunsmore [72]; Rodriguez and Tsai [207]). Indirect cost savings can also be made, given the more efficient allocation of resources that may be performed based on early metric analysis (Han *et al.* [110]).

DeMarco ([68]; [69]) suggests that the attributes of an implemented system are directly related to the characteristics of that system's input model. Given the highly structured and semi-formal nature of several widely used specification models, it is suggested that useful and consistent quantitative information relating to the system function may be derived from these representations. Currently, very little is known about analysing the complexity of requirements specifications. Nejmeh [189] suggests that this is due in part to the only recent emergence of formalised notations for specification and design tasks. As the acceptance of these notations becomes more widespread, however, assessment based on these formalised techniques will be more frequently used to improve the quality of emerging products (Fenton and Kaposi [80]).

The utilisation of graphic system models, such as those used in many specification techniques, also provides concise yet comprehensive representations of reality; this enables inexpensive analysis of essential system aspects to be performed, without having to cope with excessive internal detail. A graphic model also provides several other benefits—a common view is created for all involved in the development process, it is generally more easily refined, and it can provide the basis for quantitative indications of the scope and complexity of the project at hand (DeMarco [68]; Ramamoorthy *et al.* [200]). Goering [99] remarks that CASE is still in its infancy, but that analysis and design tools of the type described above are already available, providing the facility for automated development of high level graphical representations of systems' processing and data requirements (Grady [101]). With the further development of integrated CASE environments, the automatic extraction and application of metrics derived from specification representations is a possibility that should be fully explored.

## 2.3 Software Specification Techniques

Two of the most widely used modelling notations for the specification of commercial software requirements are the entity relationship diagram and the data flow diagram. These representation methods are also semi-formal and they have been incorporated into a number of automated development tools. They would therefore

appear to be ideal as appropriate representations for early-phase complexity analysis. The purpose of the following three sections, then, is to provide an introduction to the terminology, concepts and use of entity relationship and data flow diagrams in the development of software systems. They are not, nor are they intended to be, exhaustive reviews of the techniques themselves. This examination is then followed by a short discussion of several other common requirements specification methods. Although not as widely used as entity-relationship and data flow diagrams, they still provide structured descriptions of requirements and should therefore be considered as candidates for functional complexity analysis.

### 2.3.1 Entity Relationship Diagrams (ERDs)

Data analysis and data modelling are almost always among the first tasks performed in commercial software development. Particularly in the commercial systems domain, data modelling is often considered to be more important than process analysis because processes within a business enterprise may vary over time, whereas the central data upon which those processes operate tend to remain relatively constant (Bowker [29]; Eglington [73]).

The entity relationship (ER) modelling technique was initially proposed by Chen [45] as a method that would enable developers and users of information systems to attain a unified view of their data. Many extensions to the original model have been subsequently proposed (for example, see Chen [46], Spaccapietra [229] or March [171]), although the principal theory remains intact. The ER model has two main functions—it should provide a rigorous basis for database development and it should also serve as an accurate and understandable communication tool for analysts and users (Firns [86]; McFadden and Hoffer [176]). Two forms of entity relationship modelling are considered here:

- the Chen method [45], which uses explicit relationships
- the Finkelstein approach ([84]; [85]), which adopts an implicit relationship view.

As the function that these techniques perform is the same, many of the concepts are overlapping.

An object or entity is a real-world phenomenon about which an organisation would like to store information. For example, it may be a customer, a student, a room, or an event. An entity-set, or object set, is a collection of similar entities. The properties of entities in which an organisation is interested are called attributes. Any entity-set can be described by way of a table. For example, Table 2.1 is a representation of the STUDENT entity-set. Each row of the table corresponds to an individual entity (a student) and each column corresponds to an attribute of the entity-set. Thus the attributes of the STUDENT entity-set in this example are STUDENT-NO, STUDENT-SNAME and STUDENT-INIT.

| STUDENT-NO | STUDENT-SNAME | STUDENT-INIT |
|---|---|---|
| 850144 | Matthews | S T |
| 863289 | Smith | M F |
| ⋮ | ⋮ | ⋮ |

Table 2.1: Student entity-set

Clearly the data objects in an organisation will be related to one another—for example, a student studies a number of papers. The purpose of ER modelling, then, is to provide a concise representation of the relationships between the entities that exist in an organisation. Each relationship has two distinct properties: participation and cardinality. The participation of objects in a relationship may be optional (O) or mandatory (M). For example, a particular course of study *must* be associated with a specific student, whereas a specific paper may or may not be part of a certain student's course. Cardinality specifies the number of relationships in which an entity may participate. Generally, cardinality may be one to one (1:1), one to many (1:$n$) or many to many ($n$:$m$). As an example, a single co-ordinator may be in charge of a number of papers (1:$n$), but certain papers will have only one co-ordinator ($n$:1). Similarly, a number of students may take any combination of a selection of papers ($n$:$m$). Figure 2.1 provides a simple illustration of the two modelling techniques, including participation and cardinality considerations.

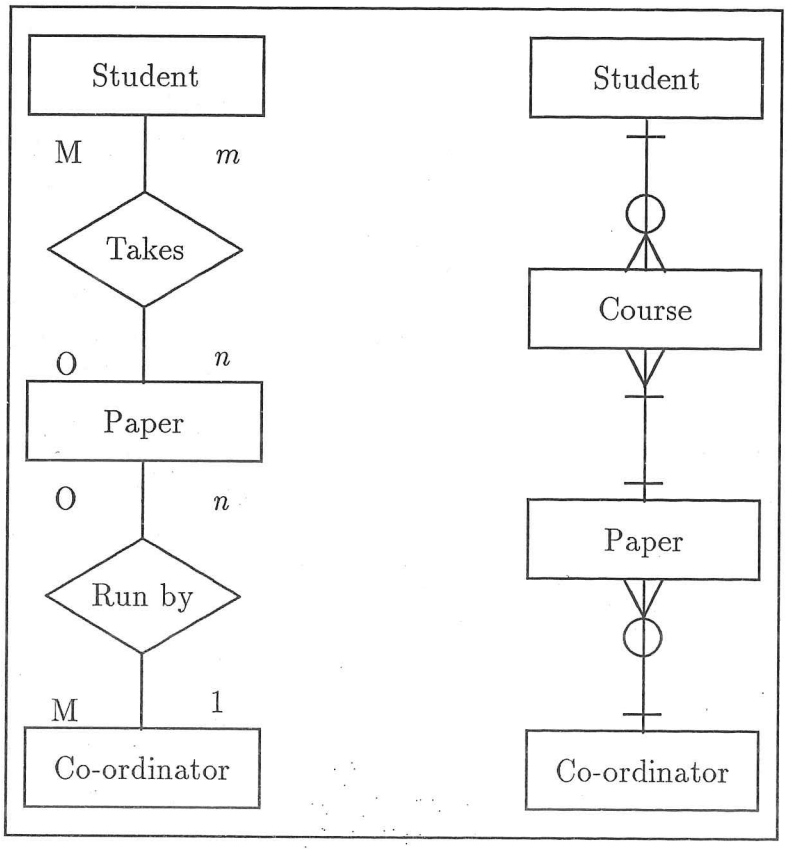



Figure 2.1: ER modelling notations

The left-hand diagram in Figure 2.1 has been drawn using an extended version of the Chen approach [45]. The rectangles denote entity-sets, the diamonds relationship sets. Cardinality is represented on the right side of the connectors (1:$n$, $n$:$m$) and participation on the left. The right-hand diagram is the same data model under the Finkelstein notation. Symbols, rather than letters and numbers, are used to represent the cardinality and participation of the relationships. Cardinality is illustrated by the use of single and diverging connector ends known as 'crow's feet', with the diverging connection representing a 'many' relationship. The bars and circles are used to reflect the degree of participation, with the bar denoting mandatory participation and the circle, optional. In corresponding relational data structures, each entity-set will be a table and each relationship will be a foreign key in the 'many' component (Date [63]) under the Finkelstein approach. Under Chen's method, some relationships will be represented by tables and others by foreign keys. In database management system (DBMS) implementation, indices may be used to represent the relationships. (Throughout the rest of this study the terms 'entity' and 'relationship' are used for convenience to represent entity-sets/object sets and relationship sets respectively.)

Since its development during the 1970s, support among both practitioners and academics for the use of ER modelling has become widespread (Choong and Churcher [49]; Firns [87]). It is currently the most widely accepted technique for logical data representation in transaction-based systems (Choong and Churcher [49]; Kilov [145]; Crozier *et al.* [58]) and it is still growing in popularity (McFadden and Hoffer [176]). This is due to a number of factors, relating to the simple yet powerful system representation that the method provides (Ferg [82]; Modell [179]). Bushell [38] asserts that a major strength of the data model is its early determination of the files that will be needed in a system. The models are developed independently of the physical structure in which the data are to be stored (McFadden and Hoffer [176]; Firns [87]); the relational model, however, has become increasingly popular as the chosen implementation structure (Choong and Churcher [49]). According to Firns [87] and Lloyd-Williams and Beynon-Davies [163], most commercially available DBMS are based on the relational model. Furthermore, relational databases can be derived directly from rigorously developed ER diagrams (Dawson and Purgailis Parker [66]; Firns [87]).

Extensive use of the ER model in determining data requirements is likely to continue in the foreseeable future, especially when it is considered that a large number of automated development environments have adopted the ER modelling convention as the basis for their data repositories (McFadden and Hoffer [176]). For example, ERMA, Teamwork, IEW/ADW, ProKit Workbench, Data Modeller, Software through Pictures, Excelerator, Blue/60, ER-Designer and IRMA all use the ER model in the derivation of data specifications for software systems. Thus the suggestion that the ER model is an appropriate early system representation that could be used as a basis for complexity analysis seems justified.

### 2.3.2 Data Flow Diagrams (DFDs)

The structured analysis methodology proposed by DeMarco [67] includes a procedure that enables the iterative identification of the activities to be performed by a system, the external entities that interact with that system, the logical data stores in that system and the various data flows between all of these components. One of the most widely used top-down approaches for the depiction of a system in these terms is the data flow diagram (DFD) (Shoval and Even-Chaime [226]; Roman [209]).

There are two main notations for hierarchically depicting the flow of data through a system. These are the Yourdon/DeMarco notation (DeMarco [67]) and the Gane and Sarson notation [93] (see Figure 2.2). Figure 2.3 is a small example of a DFD developed under the Gane and Sarson approach, depicting the fictitious costing and accounting processes of a small manufacturing company.

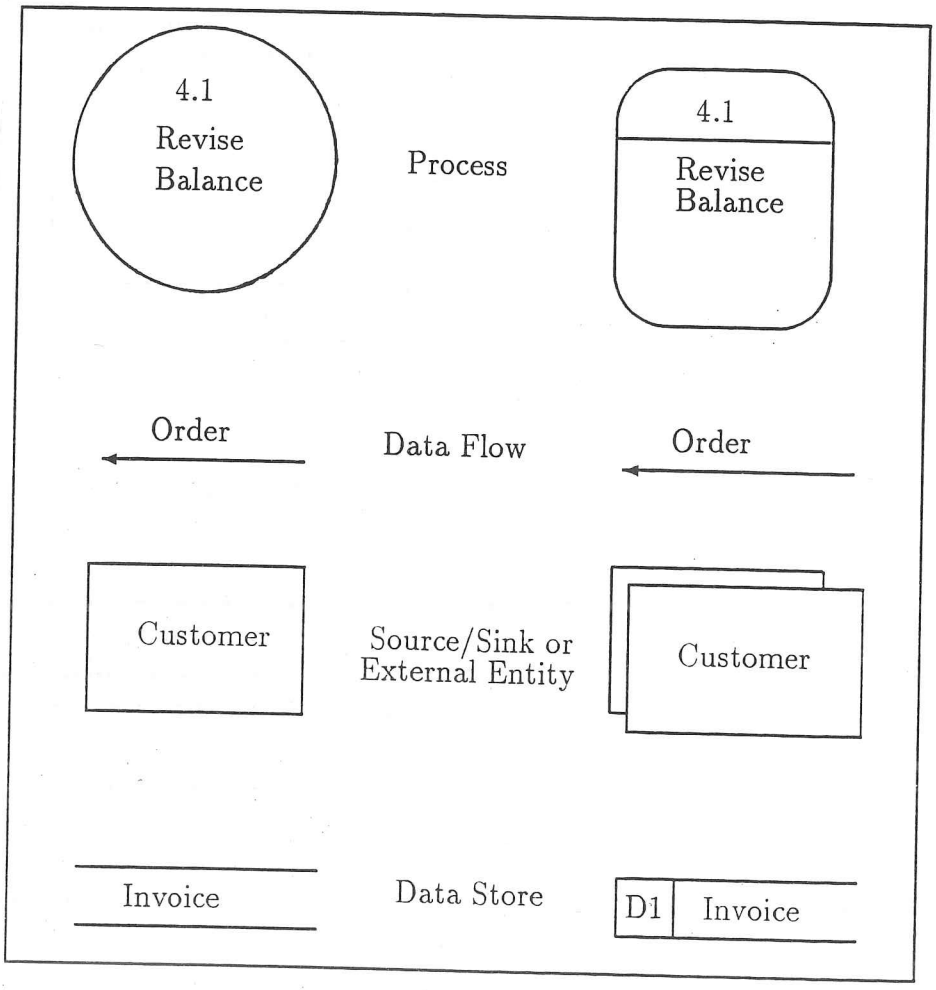



Figure 2.2: DFD modelling notations

Each element in a DFD is labelled for identification purposes—some are also numbered, depending on the notation method adopted. Processes depict a system activity performed on one or more inputs to produce one or more outputs. All of these inputs and outputs are represented as flows of data elements. The interacting external entities either supply data for use in the system or consume data produced

by it. The data stores represent repositories that may be written to or retrieved from by the processes.

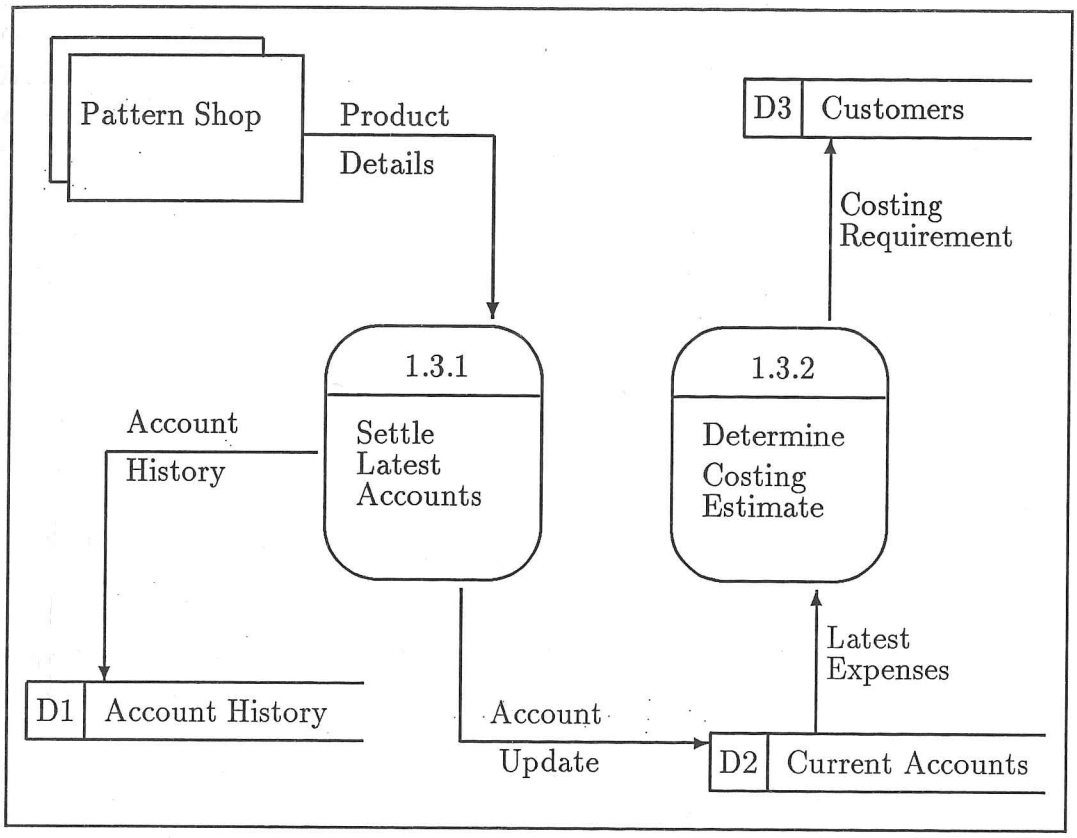



Figure 2.3: DFD example under the Gane and Sarson notation

Any transaction-based system may be represented by a network of hierarchical DFDs, based on the top-down decomposition of system processes. Usually an overview or context diagram is developed first, depicting the system in terms of its external environment—this should be similar to a high-level functional decomposition. Further partitioning normally includes the development of one second-level diagram and then any number of lower level diagrams, until all processes are elementary (DeMarco [68]). Hence in large systems, a single function can be exploded many times over until an adequate amount of detail is described and analysts are able to understand the processes fully (Senn [218]). This is normally achieved when the processes are at a level where they describe detailed computation (Keuffel [143]; Hawryszkiewycz [117]). Senn [218] therefore suggests that processes for exception and error handling should only be shown below the second or third level diagrams. Labelling of lower level processes follows a convention of decimal place addition. For example, if a second level process that is labelled as process 2 is exploded to three more processes at level three, these processes would be labelled 2.1, 2.2 and 2.3 respectively. Similarly if process 2.3 was exploded to two lower-level activities, these would be denoted as 2.3.1 and 2.3.2.

As the DFDs provide a logical or functional model of the system, procedural con-

trol is not depicted (Senn [218]; Eisenbach *et al.* [74]). Similarly, timing is irrelevant at this level; clearly processes may occur in different sequences at different times in an organisation's operations (Keuffel [142]). There is also normally no consideration of physical equipment or procedures. Although DFDs were first described in the 1970s they are still used extensively in commercial and scientific software development, as part of the structured analysis methodology. This methodology is well known and widely employed (Shoval and Even-Chaime [226]; Karimi and Konsynski [135]), with the use of DFDs being central to this technique (Hawryszkiewycz [117]; McFadden and Hoffer [176]; Hsu [122]; Tse and Pong [249]).

Apart from supporting the popular hierarchical problem-solving process, DFD exploding also improves the readability of a representation. "One ought to be able to look at a DFD and, from it, understand what the system is doing." (Hawryszkiewycz [117], p 82). Rigorous decomposition of the diagrams is also consistent with the subsequent use of modularity and structured programming in achieving easily comprehensible, cohesive and maintainable code (Hawryszkiewycz [117]; Tse and Pong [249]). As well as serving as an effective communication tool, DFDs are a significant aid in the development of processing logic (Godwin *et al.* [98]; Benwell *et al.* [17]). Direct transformation from DFDs to module structures has been examined by Tsai and Ridge [247] and Karimi and Konsynski [135] and the direct execution of DFD representations has also seen some discussion (Tate and Docker [241]; Eisenbach *et al.* [74]; Keuffel [144]).

Given the increasing use of automated assistance in software development, facilities for direct system generation from abstractions such as DFDs should become more widespread. There are already a large number of commercially available tools that employ data flow techniques (for example, Aut2, Prosa, Teamwork, Excelerator, ProKit Workbench, IEW/ADW). Most include features that assist in the development of robust specifications; for example, the enforcement of consistent element definitions, the detection of duplicate names, process balancing and co-ordination with the data repository. Processing requirements as depicted in DFDs may contribute to the overall functional complexity of a complete specification. Thus any assessment of complexity should consider aspects of the process model.

### 2.3.3 Data Analysis and Data Flow Modelling Combined

Specifications include both data and processing requirements, so both should be considered in an assessment of complexity if this assessment is to be comprehensive. However the two discussions above are generally exclusive; that is, the methods for data and process modelling are quite distinct and interaction would appear to be minimal. Teorey *et al.* [245] suggest that the requirements analysis procedure should determine or describe:

1. the enterprise's data requirements;
2. the information needed to model these requirements;
3. the transactions that are to be performed on the data.

Many individual specification techniques address either database requirements, as in numbers one and two above, or processing requirements, number three, but not both (Keuffel [139]; Freeman [91]). Whereas this has in the past been considered to be a significant problem, recent integration of the distinct techniques using automated assistance has proved to be a successful approach. Thus the procedure of detailed requirements analysis, in the commercial domain at least, frequently consists of two areas—data analysis and processing analysis (Rosenquist [210]; Gray *et al.* [102]). This approach is illustrated by Benwell *et al.* [17] in Figure 2.4, with the implication that both data flow and data structure models be used during the analysis activity.

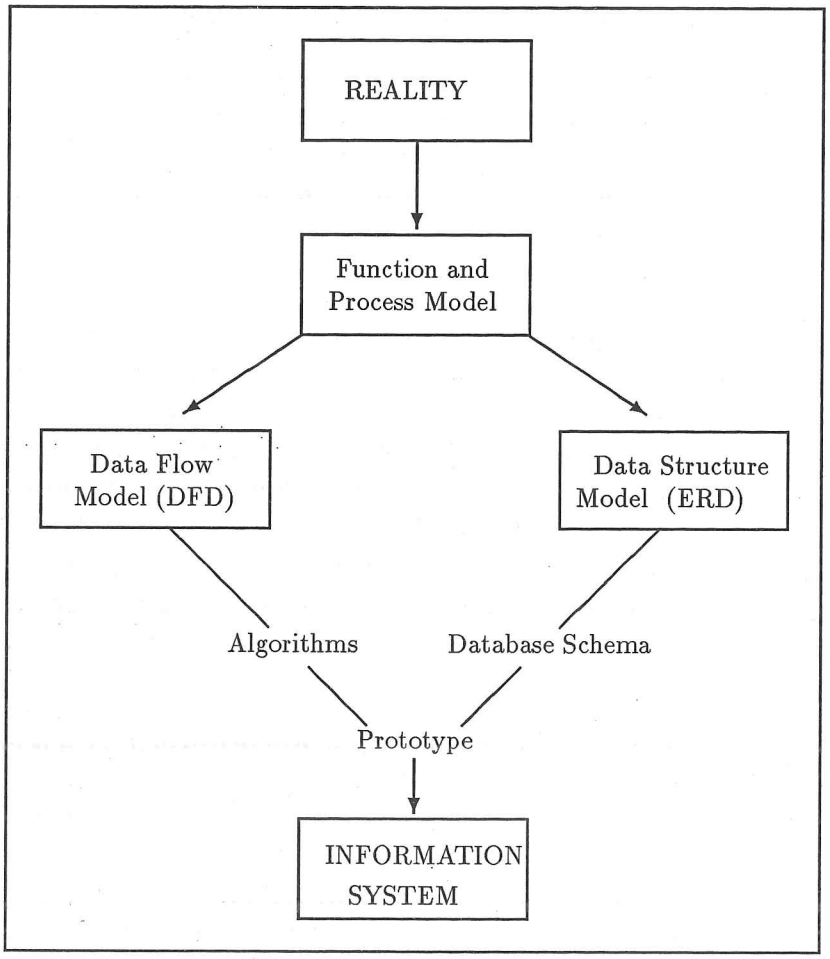



Figure 2.4: Process and data analysis in software development

A number of methodologies and authors promote the use of one technique ahead of the other. For example, Jackson [129] and McFadden and Hoffer [176] suggest that the development of process models is useful in supplementing data analysis procedures. On the other hand, Zahniser [268] and Senn [218] assert that processing analysis is enhanced by data requirements determination. Mantha [170] and Macdonald [167] suggest, in fact, that any development must use both data flow and data structure models, as an understanding of only one dimension will not result in a good final system. The question therefore arises as to which should be performed

Many individual specification techniques address either database requirements, as in numbers one and two above, or processing requirements, number three, but not both (Keuffel [139]; Freeman [91]). Whereas this has in the past been considered to be a significant problem, recent integration of the distinct techniques using automated assistance has proved to be a successful approach. Thus the procedure of detailed requirements analysis, in the commercial domain at least, frequently consists of two areas—data analysis and processing analysis (Rosenquist [210]; Gray *et al.* [102]). This approach is illustrated by Benwell *et al.* [17] in Figure 2.4, with the implication that both data flow and data structure models be used during the analysis activity.

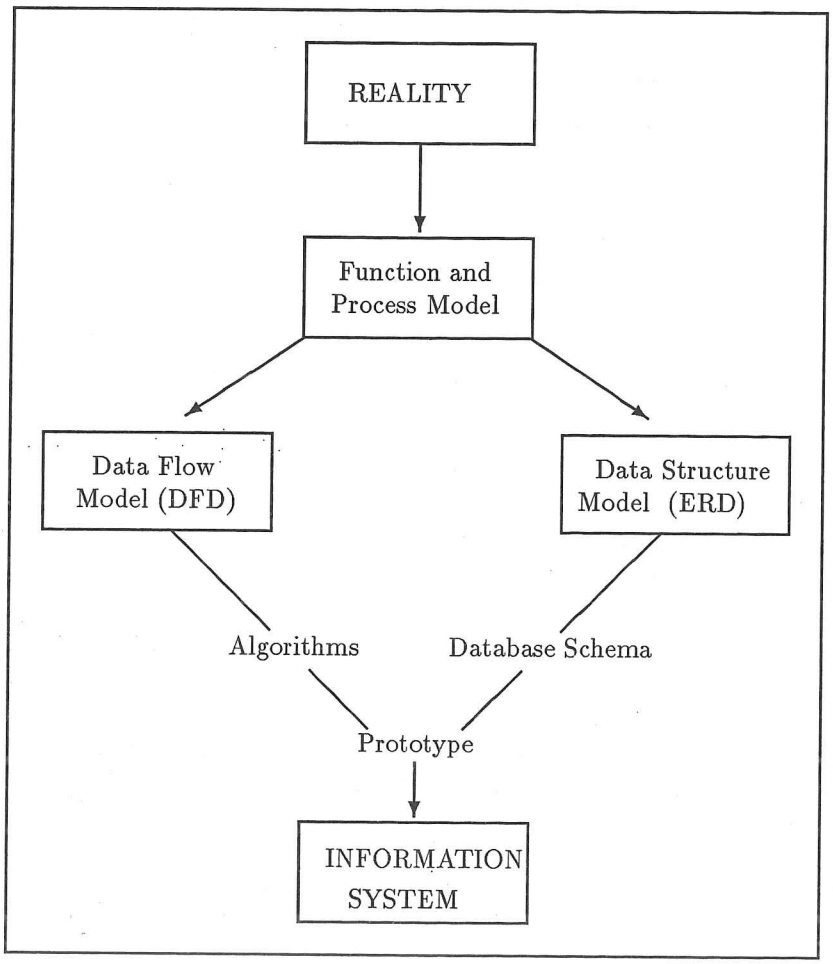



Figure 2.4: Process and data analysis in software development

A number of methodologies and authors promote the use of one technique ahead of the other. For example, Jackson [129] and McFadden and Hoffer [176] suggest that the development of process models is useful in supplementing data analysis procedures. On the other hand, Zahniser [268] and Senn [218] assert that processing analysis is enhanced by data requirements determination. Mantha [170] and Macdonald [167] suggest, in fact, that any development must use both data flow and data structure models, as an understanding of only one dimension will not result in a good final system. The question therefore arises as to which should be performed

as a first step. Bushell [38] and Hawryszkiewycz [117], however, both suggest that it simply does not matter. Achieving the overall specification is an iterative process that often takes the analyst from one technique to the other until all the details of both data and process requirements have been determined. Moreover, recent discussions have promoted the concept of relating entities to data stores so that data and process representations can be more closely linked (Harel [111]; Lee and Tan [157]). This integration will further enhance the consideration of both representations in the assessment of functional complexity.

### 2.3.4 Further Specification Perspectives

Although the data modelling and process modelling specification methods are the most widespread in the commercial systems domain, they are not totally exclusive (Tate and Verner [243]). Three further secondary specification representations are also commonly used in the description of requirements. These are the transaction and user interface representations and the functional decomposition hierarchy (FDH).

**Transaction representation** – this specification approach is popular within the database systems community. Each elementary function in a system may be considered in terms of the individual operations that it performs on single entities. For example, a process to update a customer's address may read the customer and account entities, then update the customer entity. Given adequate decomposition, low level functions or processes may be specified in this manner, providing assistance for the subsequent development of processing logic. Thus it may be considered to be a representation that combines both data and process requirements. It may therefore provide the basis for more comprehensive complexity indicators.

**User interface representation** – a perspective that is particularly applicable to development projects in which prototyping methods and 4GLs are used, given that the development of an acceptable interface can be a significant chunk of the overall effort expended in this type of environment. This representation essentially provides models of the screen and report formats that are to be subsequently implemented in the system. As interactive systems, by their very nature, use screen displays, and many transaction processing systems produce reports, a consideration of the complexity of this representation is essential if an overall assessment of complexity is to be obtained.

**Functional decomposition hierarchy** – often produced as a levelled description of the functions to be provided by a system, this representation normally illustrates the module calling structure that will eventually be generated or constructed. The number and interaction of the modules are likely to have an impact on system complexity so this representation should also be considered in any functional assessment scheme.

Thus all five specification perspectives, as depicted in Figure 2.5, are quantifiable in terms of the contribution that each may make to the overall complexity of a complete specification. Consideration of each should help to ensure that the assessment of complexity is as comprehensive as possible.

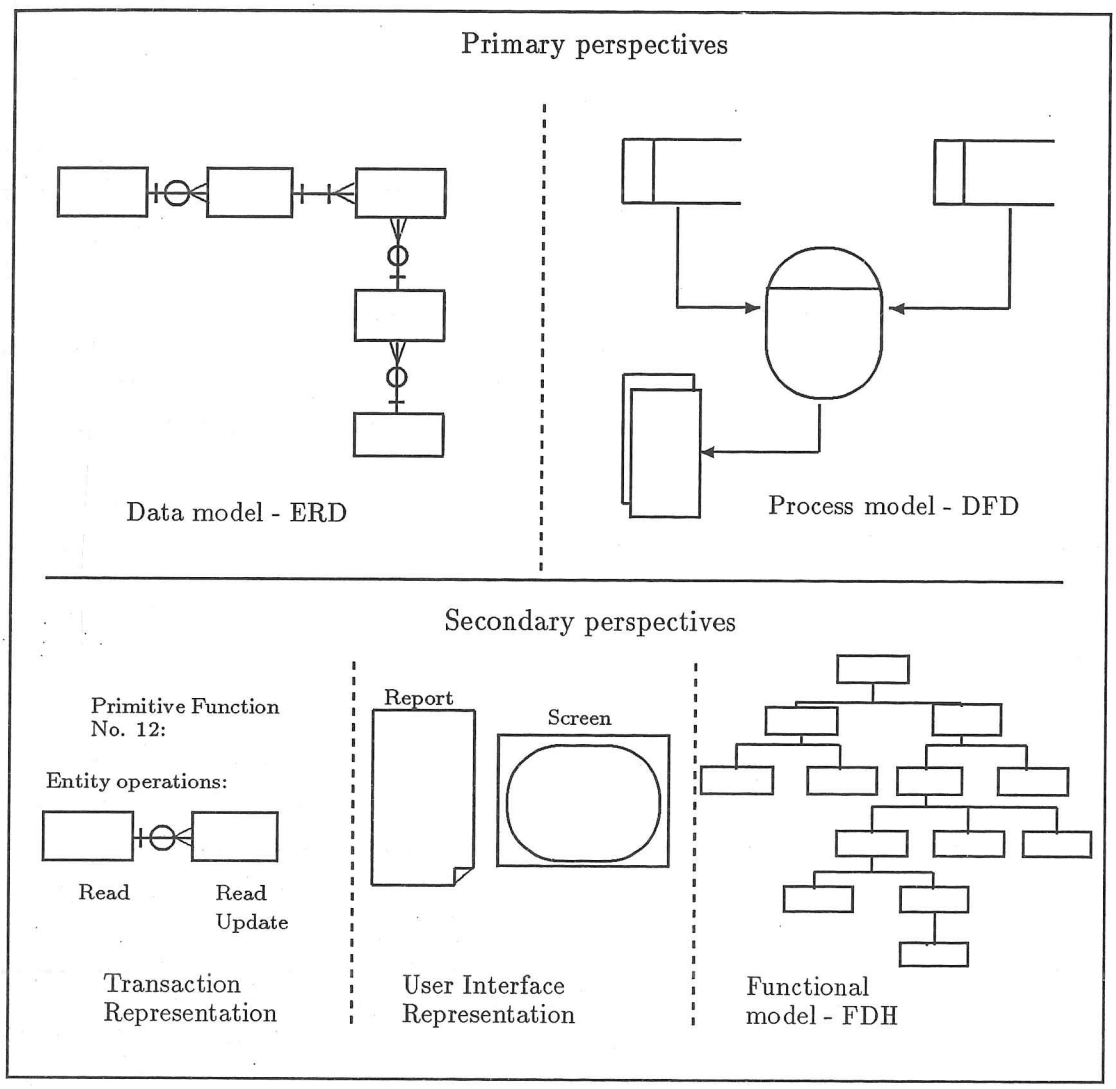



Figure 2.5: Five perspectives of a system specification

## 2.4 Development Automation

Up until less than ten years ago software development was a largely manual activity. This created a situation where both the software product and the progress of software development were significantly influenced by the people involved in a given project. Thus estimation of project management outcomes using program-based complexity indicators was fraught with difficulties, as the impact of personnel ability was so great. Recent advances in technology, however, have greatly extended the degree of automation that may be applied to software development tasks. It is

therefore suggested that this will greatly reduce the effect of specific personnel on the outcome of a project. More effective estimation should therefore be possible. In terms of early-phase automation, CASE tools are becoming increasingly accepted in mainstream data processing environments (Chen and Norman [43]; Brown and McDermid [33]).

### 2.4.1 CASE

CASE represents a comprehensive philosophy for modelling businesses, their activities, and information systems development within this environment (Gibson and Senn [96]). It is often defined as the application of automated technologies to traditionally manual software engineering and development processes. The objectives of CASE use include the improvement of development productivity and product quality and the achievement of greater control over the software development process (Case [40]; Factor and Smith [78]). The term 'CASE' itself, however, has been used more widely as a descriptive term for the vast assortment of automated development tools that have become available over the last nine years (Burkhard and Jenster [36]).

Acceptance of CASE technology has been gradual over this period, but it is now seen as an integral component of software development in the commercial systems domain (Vargo and Kong [251]; Keuffel [138]). Jones [133] suggests that the fundamental concept of the CASE approach to development is one of economics—traditional development, without automation, is simply too labour-intensive to be viable (Tate *et al.* [244]). This is particularly the case in the business sector. Thus the majority of CASE tools have been produced for use in this application area (Chen *et al.* [44]). As a generic term, the functionality of products described under the CASE banner varies widely, ranging from simple diagramming tools to integrated systems that enable an analyst to produce detailed requirement models and automatically generated systems (Williamson [261]; Haddley and Sommerville [105]; Firns [87]). It has been suggested that in time CASE tool sets will largely displace programming in the construction phase (CIS [50]). Burkhard and Jenster [36] assert, however, that CASE will only be successful if it can be applied to all development phases, as single-phase concentration will not have a significant impact on overall quality outcomes.

One of the most notable benefits of CASE use is the resulting improvement in development productivity. This is most often the reason that organisations cite for the adoption of CASE technology (Bishop and Lehman [21]; Horch [121]), and has been shown to be the outcome in a number of cases (Buckler [35]; Burkhard and Jenster [36]; Statland [232]; Snyders [227]). Another of the promised benefits that CASE provides is an increase in the quality and accuracy of the software product (Rummens and Sucher [214]; Belson and Devonald [16]). Improved quality is normally assessed in terms of reduced error occurrence and change requirements in the delivered product. Thus the real benefits of CASE use are the early detection of errors or functional inconsistencies (Glass [97]), and the ease with which these problems can be resolved in an automated environment (Rinaldi [203]; Williamson [261]; CIS [50]). This is most often achieved in CASE tools through the use of automatic

specification and design consistency checks (Forte and Norman [89]; King [148). The dual goals of productivity and quality improvement would seem to suggest that the future widespread use of CASE technology is assured:

> While CASE is not a panacea for developers' problems, it does appear that its clear focus on quality and productivity will make it the next generation environment for building and maintaining bespoke applications (King [146], p 37).

This supports the continuing use of CASE and other automated tools in software development, an issue central to this study, in that automation must be extensive if functional complexity analysis and subsequent estimation are to be effective.

### 2.4.2 Application Generators/4GLs

Whereas CASE tools have generally been developed for use in the early phases of development, the focus of 4GLs has been on the simplification of coding and testing (Norman and Chen [191]). Automation of these activities is also crucial to the development of an appropriate functional complexity analysis scheme, in that generally applicable relationships can only be established when automation is evident throughout the life cycle. Just as the term CASE covers a wide array of tools and techniques, so the term '4GL' encompasses many differing products. Also known as application generators, these database oriented product-sets normally contain some or all of the following components (Gavurin [94]):

- data entry/update screens with validation
- prototyping facilities
- non-procedural query languages
- screen painters
- report generators
- a centralised data repository
- intelligent defaulting
- code generation facilities.

4GLs evolved because of the need for increased productivity and quality in commercial software development. They offer environments where DBMS support is provided to maintain data independently of applications (Firns [87]). The languages generally enable developers to focus on the problem to be solved rather than on how they should solve it (Clarke [51]). Thus developers are relieved of lengthy implementation details (Chen *et al.* [44]; King [146]) and may therefore concentrate on ensuring correct system functionality.

One of the most widely promoted features of 4GLs is the non-procedural nature of query and report requests. For example, suppose a company report is required listing the last names of all employees on file with their unique employee number, and the date on which they started work, in alphabetical order of employees' last names. Using Cognos Inc.'s PowerHouse product, this could be achieved with the following (MacDonell [168]):

```
ACCESS EMPLOYEES
SORT ON LASTNAME
REPORT LASTNAME, EMPLOYEE-NO, START-DATE.
```

Thus the primary strength of these products is often seen to be the relatively small working set of commands that is needed to perform often complex data manipulation. 4GLs are therefore able to provide an environment for the development of systems to an operational level of user acceptance in a timely and cost-effective fashion (Sallis [215]; Alavi and Wetherbe [1]). The system prototyping approach, enabling the rapid development of mock-up systems, almost always depends on the use of a 4GL (Keuffel [144]). This procedure allows for a greater degree of communication between users and designers concerning the functionality of systems (Mason and Carey [174]; Jarke [130]), leading to decreased maintenance requirements as fewer errors and omissions are made during the early phases of development (Necco *et al.* [188]; Sumner [234]).

Due to the use of English-like structures 4GLs may also be used by personnel other than those formally trained in systems development (Senn [219]; Lin [161]; NCC [187]), often enabling a quicker response to application requirements (Lloyd's [162]). Harel and McLean [112] found evidence for the hypothesis that, based on a subjective assessment of task complexity, programmers of any skill level were more productive with a 4GL than with a 3GL, no matter what the complexity of the task. It should therefore be clear that in the development of commercial applications the need to *program* complex software can now be largely avoided, as the design and construction of the logic can be carried out relatively easily through the use of appropriate database techniques implemented in English-like 4GLs (Martin and McClure [173]). Data entry programs in particular can be generated almost automatically based on centrally stored data definitions. Similarly, simple or standard output formats can also be directly generated. Thus programmer-independent development from specifications is possible, further enhancing the opportunities for effective functional complexity analysis.

## 2.5 Data Specification, CASE and 4GLs – An Integrated Approach

Although there are certainly a number of development methodologies that do not use the above tools and techniques it would still seem reasonable to suggest that automated structured specification and development methods will continue to be used within the business community for the foreseeable future. This is due to a

number of reasons, including vendor commitment to automation, user investment in tools and methods and, to a lesser extent, tradition. This will lead to more extensive tool and method integration, resulting in the provision of a single development and maintenance environment (Jones [132]; CIS [50]; Stamps [231]). It is therefore envisaged that the proposed complexity analysis strategy, which assumes the existence of such an environment, will not become obsolete in the near future.

The integration of structured specification methods within CASE tools is very much part of today's technology. Necco *et al.* [188] conclude that structured analysis techniques will be used increasingly, due in part to the more practical and economic environment that CASE provides for them (Chikofsky and Rubenstein [48]). The data flow and data analysis methodologies appear to be particularly well supported in current CASE products (Gray *et al.* [102]; Robinson [204]), in response to perceived analyst requirements. One empirical survey of forty-six management information system (MIS) professionals highlighted a large degree of interest in assistance for the development of DFDs, code generation, specification definition and data modelling (Burkhard and Jenster [36]). Another study used multi-dimensional scaling and cluster analysis techniques to investigate the perceptions of ninety-one software engineers relating to the impact of seventeen tools and techniques on productivity. By performing 136 pair comparisons, automated DFD assistance was chosen as the most effective tool for increasing productivity over manual methods. The use of data dictionaries was the next most frequently chosen aid. ER or data modelling was ranked eighth, ahead of lower level support tools such as structure chart and structure diagram editors and record layout generation facilities (Norman and Nunamaker [192]). In another examination of twelve CASE tools, Vessey *et al.* [255] found that all of the products investigated included DFD assistants and eight of the twelve included ER modelling facilities.

The apparent popularity of DFD generation assistance is not unexpected. DFD production is an integral and often first step of most structured development methods in the commercial environment, yet without automated assistance it has been a time-intensive manual activity. This has now been largely overcome with the availability of automated tools (Case [40]; Burkhard and Jenster [36]; Chikofsky and Rubenstein [48]). In terms of data modelling assistance, the basis for the central data dictionary in CASE tools is frequently the ER model (Choong and Churcher [49]; Keuffel [138]). Turnbull [250] therefore suggests that systems developed from ER model foundations will be at the heart of future projects. The interface between a 4GL and an underlying DBMS is usually provided by a data dictionary, the basic inputs for which may be specified by ER models—ER modelling is therefore central to the development of systems using 4GLs (Firns [87]). Moreover, Sallis [216] states that many CASE tools already have an interface to the functionality of 4GLs, through the use of centralised data dictionaries. Existing application generators have therefore been extended to include graphic CASE assistance in earlier development phases (Clarke [51]). Thus 4GLs are now often referred to as lower CASE products (King [147]).

Given that this comprehensive environment is, or will be, widely used in the commercial domain, new complexity analysis schemes should intuitively be based

on aspects of the products and processes that are part of this environment. Data structure and data flow representations, widely used in the business systems domain, therefore provide the basis for many of the analysis scheme measures proposed in Chapter 4. The increasing use of automated tools in the commercial development domain would also suggest that any assessment scheme should be applicable to such an environment—the proposal therefore assumes that automated assistance in development is extensive. Automation also has a number of implications regarding software understanding, examined in the next chapter. Finally, to ensure applicability of results, the systems used in validation should also be representative of this setting; that is, they should be specified and developed using structured data representation techniques in an extensively automated environment. Hence the systems used in the validation phase of the current study, described in Chapter 5, match this classification.

# Chapter 3

# Specification-Based Functional Complexity Analysis

## 3.1 Introduction

Complexity has often been considered to be synonymous with, or at least related to, understandability; that is, software that is more complex will generally be more difficult to understand. Objectively assessing ease of understanding, however, is extremely difficult and can be easily confounded by external factors. In spite of this, a number of complexity analysis techniques have purported to measure the understandability of implementations. This chapter therefore considers the role of software understandability in an automated environment. Implementation methods, also considered to be significant contributors to complexity, are of less influence in an automated environment. The applicability of traditional implementation-based measures in assessing functional complexity is therefore also discussed. Current specification analysis methods are then critically reviewed in order to provide a basis for improvements that may be incorporated into the assessment scheme proposed in this study.

## 3.2 Software Understandability

Several complexity assessment methods have concentrated on measuring the difficulty encountered by developers in understanding software implementations. The increasing use of CASE techniques and 4GLs in business software development, however, makes the transformation from specification to final system much less dependent on implementation methods or on the styles and abilities of individual programmers (Tate and Verner [242]; see Figure 3.1 for an illustration of the transformation process (McFadden and Hoffer [176]). Relative levels of implementation understandability are therefore of less importance when it comes to assessing the ease with which software developed under an automated environment can be constructed and maintained.

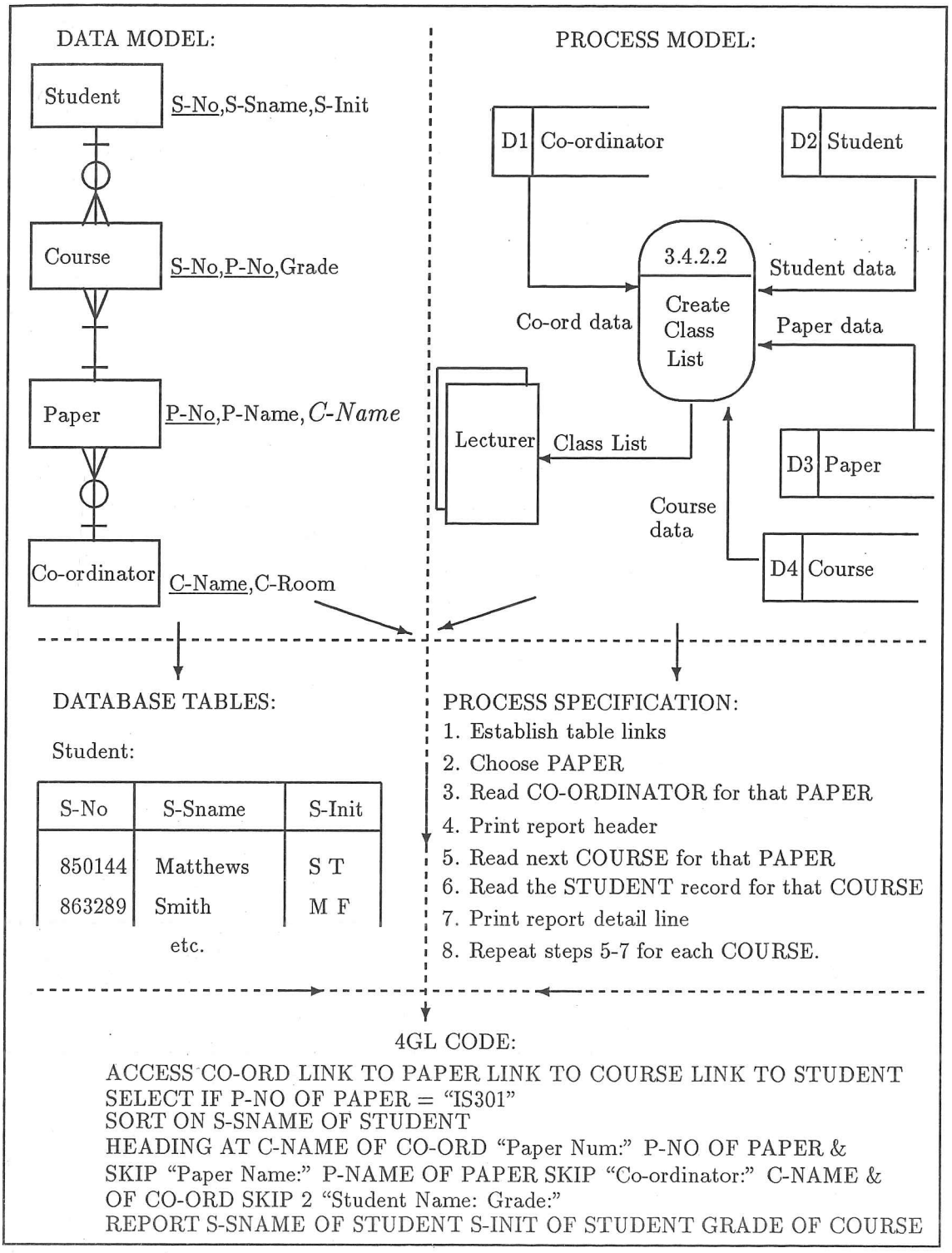



Figure 3.1: Transformation from data and process models to 4GL code

A number of specific complexity measurement techniques have been said to quantify the understandability of software systems with the objective of relating understandability to effort requirements or to the likelihood of errors (Evangelist [76]; Bastani [12]; Harrison and Magel [115]; Chapin [41]). This claim is generally unjus-

tified, as understandability is by its very nature dependent on the individual who is attempting to understand the software—it is clearly more probable that a student programmer would have significantly more difficulty comprehending a system than would an experienced developer. Understandability is therefore relative and cannot be effectively measured without including some consideration of the individual programmer involved. The assessment of programmer ability has its own problems, however—who is to say that two programmers who have worked with the same language in the same environment for the same length of time will have the same ability to comprehend software? Generalisation of results derived from individuals over the development population is therefore often difficult, if not impossible. This is not a totally fruitless situation, however. The fact is that development managers are not interested in the level of understandability *per se*; rather they are concerned with the impact that complexity has on other process and product attributes.

As previously discussed, it has been widely suggested and generally accepted that a more complex system will indeed be more difficult to understand (Harrison and Cook [114]; Lew *et al.* [159]; Rodriguez and Tsai [207]), and it is therefore in this area where much of the previous complexity assessment work has been focused. Operational measures provided to reflect this difficulty of understanding have been based on a diverse selection of factors, for example, the number of errors that are found during development, development effort, or the number of post-delivery modifications required. It is, however, exactly *these* factors in which the project manager's interest lies. The fact that *system 1* may be one-third as difficult for *programmer z* to understand as *system 2* is of little practical use to the manager. What he or she wants to know is how the fundamental complexity will affect, say, the overall development time estimates, or the current predictions for testing requirements. Furthermore, quantifying levels of system understandability has also become less important with the use of integrated CASE tools and 4GLs (Nelson [190]), particularly in the development of new transaction processing and data retrieval systems. Once a specification has been completed, direct system generation reduces the need for interpretation from the specification to the coded product (Kerr [137]; Lin [161]; Crozier *et al.* [58]). It is therefore suggested that an analysis of functional complexity based on software specifications can be performed without regard for 'understandability', to provide direct and independent estimates of system development attributes.

## 3.3 Functional Complexity versus Implementation Complexity

Just as all systems written or developed under a particular language or method are directly affected in terms of complexity by that method, so too are they directly affected by the function that they are to perform. That is to say, a piece of software to perform a given function will have an inherent degree of complexity, regardless of the chosen method of implementation. Under a largely automated environment, then, functional complexity is likely to be much more significant than implemen-

tation complexity. Most previously proposed complexity measurement approaches, however, are based solely on implementation complexity, as discussed below.

- Lines of code-type measures are clearly from the implementation-based category; they make no consideration of what it is that the lines do, only how many there are. Hence, two distinctly different systems, in terms of what they do, can have the same level of 'complexity' under these counting schemes. Furthermore, lines of code are now a far less visible software product due to the use of automated development environments.

- Software science metrics (Halstead [107]). Although Halstead attempted to incorporate psychological aspects into his measurement family, all of the individual measures are based, in part at least, on the frequency of operators and/or operands. These counts are again directly dependent on the method of implementation. The only possible exceptions are the potential volume and derived measures (program level and others). The accuracy and derivability of potential volume has, however, been questioned (Naib [185]; Hamer and Frewin [108]), placing some doubt over the applicability of these measures. Furthermore, the underlying psychological models employed by Halstead have also been widely criticised (Coulter [56]).

- Cyclomatic complexity (McCabe [175]), said to be typical of topological measures, is also implementation-based. By its very nature, code topology is a direct result of the methods used in software construction, and may be distinctly different for the same functional system implemented in various ways. The metrics are also inherently inapplicable to most 4GL-type code as control flow is no longer explicit.

- Structural design metrics are clearly derived from a higher level of development. Many, however, appear to be based on the low-level (close to physical) design of a system and therefore consider control and data flow between physical modules. This may be problematic, in that the design phase is often performed with a particular implementation method or physical structure in mind. So once again the measure of complexity is founded in the physical rather than the functional, although not to the same extent as in the previous techniques.

- Functional metrics currently cited and in use are following a promising line but often treat complexity in a superficial way, as part of the means to what are considered to be greater ends, that is, size and productivity estimation.

All this is not to say, however, that implementation complexity should always be disregarded. For example, depending on the way in which maintenance is performed, the method of implementation may have a significant impact on the ease with which software can be changed. Maintenance has traditionally been carried out based on existing code. It is well recognised, however, that code tends to become more complex as it is changed. Programmers therefore become less successful in determining the underlying logic from the code, and so the effectiveness of maintenance

decreases over time. In this environment traditional metrics may be used effectively to assess the current code complexity. With the use of comprehensive automated assistance, however, maintenance should begin to follow a cycle similar to that of new development (CIS [50]). That is, changes should be made to the relevant parts of the original specification model (Choong and Churcher [49]; Harel [111]; Baxter [14]), with the corresponding parts of the final system then being regenerated by the tool based on the revised specification. Naulls [186] remarks that vendors of code generators already advise that changes should not be made to their code unless absolutely necessary. It is recommended that new code be regenerated from new input models (Chen and Norman [43]). Therefore the functional complexity of existing specification structures may also become more important for maintenance tasks (Maria [172]). Thus for both systems development and systems maintenance activities, particularly within the MIS domain, it is suggested that functional complexity analysis, as determined from a system's specification, is now of prime importance.

## 3.4 Specification Analysis Research

With the widespread use of structured specification methods and the increasing impact of development automation, opportunities for direct and objective specification-based complexity assessment are becoming increasingly viable. As yet, however, specification-based assessment methods are not widely used. As discussed in the previous chapter, a number of specification methodologies promote the determination of logical or conceptual data structures as one of the first steps to be performed in the development process. Hence, there has recently been increased support for the quantitative analysis of this system abstraction. Davis and LeBlanc [65] suggest that measures based upon a system's data structure could be used to evaluate the requirements and the design, through mappings to data dictionaries and data flow diagrams. Processing requirements of the system can also be determined, given the assumption that a well structured function should correspond to the underlying logical data structure (Jackson [128]). Once the data has been specified, then, the properties of the data structures can play an increasingly important role "...since they are the foundation of the final implementation." (Tsai *et al.* [248], p 240).

The attention of complexity researchers has only recently been turned to data-oriented analysis. Demurjian and Hsiao [70] and Shoval and Even-Chaime [226] provide discussions on comparing the complexity of various data modelling representations, that is, relational, network, flow-based, data-based, etc., but offer no methods for evaluating actual specifications developed in a given model. Blaha *et al.* [22] suggest that the relative merits of a specification's data model can be measured by:

1. performance – this relates to speed of access to data elements;
2. integrity – relating to the likelihood of data accuracy;
3. understandability – relating to the coherence of the model to users, other designers and the original designers after a period of time;

4. extensibility – relating to the ease of extension to incorporate new applications without disruption.

Webster [258] suggests a similar set of criteria, including conciseness, clarity and naturalness, for comparing the conceptual complexities of information representation techniques. Although these are all certainly attributes of interest, no quantitative assessment methods have been provided.

Batini *et al.* [13] remark that, to their knowledge, there are no quantitative and objective measures of conceptual understandability that can be applied to data models. They go on to suggest the consideration of aesthetic aspects, such as the shape of a diagram or the number of line crossings, as a possible assessment method. For further discussion on the comparative complexities of diagram layouts, see Protsko *et al.* [199] or Tan *et al.* [238]. In a discussion of work-product measurement, Grady [101] suggests the use of writing analysis techniques for text assessment and DeMarco's Bang [68] for DFDs, but no mention is made of any data modelling work-product at the specification stage. A data dictionary is introduced as a work-product of the design phase but no complexity analysis method is provided with it. Grady [101] remarks that very little metrics research has been centred on the data aspect of systems.

Yet the data structure of a system is said to be a critical factor in influencing final product complexity (Tsai *et al.* [248]). It would therefore seem to be beneficial to derive quantitative specification measures based at least in part on some representation of the data structure. Measures taken from this product would be available almost from the beginning of development and the functionality and structure of the subsequent programs should follow very closely the structure of the data that they manipulate—"...Thus, the more complex the data structure is, the more complex the program will be, the more difficult it will be to maintain that program." (Tsai *et al.* [248], p 241; Harel [111]). Low-level data structure measurement, that is, comparative complexity assessment of pointers, linked lists and arrays, has been investigated by Tsai *et al.* [248] and Iyengar *et al.* [127]. Higher-level dynamic and static analysis of database schemas has also been described by Gerritsen *et al.* [95]. Techniques for conceptual data structure assessment, however, have undergone very limited discussion.

High-level process determination is also one of the first tasks to be performed in development projects. As this procedure has been an integral component of many methodologies for a number of years, one might have expected that structural assessment methods for process representations would have been well developed by now. However, due to the absence of automated assistance, measurement until recently has only been carried out on lower-level descriptions, for example, structure charts or module calling trees. The use of CASE technology, however, now provides capabilities for the simple assessment of data flow representations, and also ensures that this high-level abstraction is directly related to the final system through facilities for direct system transformation. As these have been relatively recent advances, work on data flow complexity analysis is not yet widespread.

### 3.4.1 Current Specification Analysis Approaches

DeMarco [68] suggests that development effort is a function of a system's information content. He further asserts that the information content of a final coded system is a well-behaved function of the information content of that system's specification. Unfortunately the lack of uniformity among specification structures, he continues, prevents direct information theory evaluation of the traditional documents—however, he does suggest that the use of standard specification models could provide a consistent framework for structural comparison. In essence, this provides the basis for the development and use of functional measures. A number of existing techniques are now discussed in order to determine the desirable characteristics of new assessment schemes. Although several of these existing measurement methods have size or productivity estimation as their goal, they all attempt to consider system complexity in some way. It should be kept in mind, however, that the somewhat superficial treatment that complexity is given under some techniques may not take away from the overall credibility of the methods in achieving their intended purpose. Moreover, the final goals of these approaches are not far removed from the goals of complexity assessment, particularly in terms of effort estimation.

#### Function Point Analysis (FPA)

Function point analysis (Albrecht [2]) is the most widely investigated of the function-based approaches. Quantification of complexity under this technique is performed as a sub-task of the complete model, the overall original purpose being the determination and prediction of development productivity. Each system is considered in terms of the number of inputs, outputs, inquiries, files and external system interfaces that it contains. The system total for each of these attributes is multiplied by a weighting factor appropriate to its complexity in the system (simple, average or complex), based on the number of data elements and/or file types referenced. The combined total of all of these products is then adjusted for application and environment complexity—this can cause an increase or decrease of up to 35% in the raw function point total. Calculation of the adjustment factor is carried out by considering the need for certain features in the system, for example, distributed processing and ease of installation. Each of the fourteen factors is assigned a degree of influence of between zero (no influence) and five (strong influence), and these are summed to give a total degree of influence, denoted $N$. One of the fourteen factors is allocated for the consideration of complex processing. A technical adjustment factor is then calculated as $(0.65 + 0.01(N))$. This adjustment factor is subsequently multiplied by the raw function point total to determine the final function point value delivered by the system. According to Grupe and Clevenger [104] the underlying assumption of FPA is that higher numbers of function points reflect more complex systems; these systems will consequently take longer to develop than simpler counterparts.

Complexity is therefore considered in two ways during the analysis. It is questionable, however, whether this consideration is completely adequate. Albrecht acknowledges that the complexity weights applied to the raw function point counts were "...determined by debate and trial." (Albrecht and Gaffney [4], p 639). The

### 3.4.1 Current Specification Analysis Approaches

DeMarco [68] suggests that development effort is a function of a system's information content. He further asserts that the information content of a final coded system is a well-behaved function of the information content of that system's specification. Unfortunately the lack of uniformity among specification structures, he continues, prevents direct information theory evaluation of the traditional documents—however, he does suggest that the use of standard specification models could provide a consistent framework for structural comparison. In essence, this provides the basis for the development and use of functional measures. A number of existing techniques are now discussed in order to determine the desirable characteristics of new assessment schemes. Although several of these existing measurement methods have size or productivity estimation as their goal, they all attempt to consider system complexity in some way. It should be kept in mind, however, that the somewhat superficial treatment that complexity is given under some techniques may not take away from the overall credibility of the methods in achieving their intended purpose. Moreover, the final goals of these approaches are not far removed from the goals of complexity assessment, particularly in terms of effort estimation.

#### Function Point Analysis (FPA)

Function point analysis (Albrecht [2]) is the most widely investigated of the function-based approaches. Quantification of complexity under this technique is performed as a sub-task of the complete model, the overall original purpose being the determination and prediction of development productivity. Each system is considered in terms of the number of inputs, outputs, inquiries, files and external system interfaces that it contains. The system total for each of these attributes is multiplied by a weighting factor appropriate to its complexity in the system (simple, average or complex), based on the number of data elements and/or file types referenced. The combined total of all of these products is then adjusted for application and environment complexity—this can cause an increase or decrease of up to 35% in the raw function point total. Calculation of the adjustment factor is carried out by considering the need for certain features in the system, for example, distributed processing and ease of installation. Each of the fourteen factors is assigned a degree of influence of between zero (no influence) and five (strong influence), and these are summed to give a total degree of influence, denoted $N$. One of the fourteen factors is allocated for the consideration of complex processing. A technical adjustment factor is then calculated as $(0.65 + 0.01(N))$. This adjustment factor is subsequently multiplied by the raw function point total to determine the final function point value delivered by the system. According to Grupe and Clevenger [104] the underlying assumption of FPA is that higher numbers of function points reflect more complex systems; these systems will consequently take longer to develop than simpler counterparts.

Complexity is therefore considered in two ways during the analysis. It is questionable, however, whether this consideration is completely adequate. Albrecht acknowledges that the complexity weights applied to the raw function point counts were "...determined by debate and trial." (Albrecht and Gaffney [4], p 639). The

absence of empirical foundation for these weights has since received criticism from several quarters (Roland [208]; Shepperd [222]; Arthur [6]). Moreover, with respect to the raw counts, the categorisation of the system components as simple, average or complex, although clearly straightforward, seems to be rather simplistic in terms of a comprehensive assessment of complexity—Symons [235] provides the example that a component consisting of over 100 data elements is assigned at most twice the points of a component that contains just one data element. It is also suggested that the weightings are unlikely to be valid in all development situations.

There are similar problems with the technical complexity adjustment process. It would seem unlikely that the consideration of the same fourteen factors would be sufficient to cope with all types of applications. Also, adjustments to the raw counts can only be affected by a factor within the zero to five range which, although simple, is unlikely to be appropriate in all cases. Consideration of processing complexity in only one of the fourteen factors is not only inadequate, it may also not be practically applicable at the software specification stage. It is recommended that the value of the adjustment factor for complex processing should be based on a number of factors, including the need for sensitive control/security processing and extensive logical or mathematical processing (Rudolph [213]; Albrecht and Gaffney [4]; Gordon Group [100]). It would seem unlikely, however, that information of this kind would be available at the conceptual modelling stage. This reinforces another drawback of the method, in that it is not based on modern structured analysis and data modelling techniques (Tate and Verner [243]).

Overall, then, the technique tends to underestimate systems that are procedurally complex and that have large numbers of data elements per component (Symons [235]; Verner and Tate [253]). Shepperd [222] and Ratcliff and Rollo [202] also remark that the identification of the basic components from the specification can be difficult and rather subjective—different analysers may therefore use different logic to determine the number and complexity of the functions provided by the system (Rudolph [212]; Conte *et al.* [54]). This subjective element can dominate the final results, reducing the utility of a seemingly quantitative process (Symons [235]; Wrigley and Dexter [265]).

### MARK II FPA

Symons ([235]; [236]) has developed a specification-based sizing and effort estimation technique based on a revised version of the function point analysis method. He identified several failings with Albrecht's original technique, as outlined in the previous section, pertaining particularly to the classification and weighting strategies used in the original theory. Symons [235] further suggested that these problems were compounded by technology-driven changes, so that, for example, the original concept of a logical file was no longer appropriate in the database environment that now dominates business systems. Symons [235] therefore adopted the entity type as the basic data equivalent for transaction-centred systems.

The MARK II method involves the identification of all the inputs, outputs and processes associated with each externally triggered logical transaction performed

by a system. To assess the size contribution of the input and output components, Symons' method [235] counts the number of data elements that are used in and produced by the transaction. This is founded on the assumption that the effort for formatting and validating an input or an output is proportional to the number of data elements in each. Symons [235] suggests that this provides greater objectivity in the counting procedure when compared to Albrecht's somewhat subjective approach.

Identification and evaluation of the process component is more difficult, in terms of developing an appropriate size parameter for this aspect of a transaction. The method suggested by Symons [235] relies on previous work on internal structure measurement based on code branching and looping (McCabe [175]). It is suggested that the data structure employed by a system may provide a basis for the assessment of processing complexity. At the specification stage, this is represented by the access path of a transaction through the system entity model. Symons [235] states that since each step in the path correlates to a branch or a loop, the processing complexity will be directly related to the number of entities referenced by the transaction. Although this argument was originally considered to be rather tentative, providing only a crude measure of processing complexity, it has remained intact and has been reinforced in Symons' more recent work [236].

The formula for the raw size factor in unadjusted function points is therefore calculated by multiplying locally calibrated weighting factors with the basic counts of input and output data elements and the number of entity references in the system, and then summing together the three weighted totals for all of the system's transactions. An industry standard set of weightings is available as a starting point. The technical complexity adjustment procedure is very similar to that of the original theory except that the fourteen Albrecht factors [2] are augmented by five or more new characteristics.

Using counts of data elements for the input and output components is a positive and more contemporary approach, as is the adoption of entity-based assessment. Under this method, however, there is no consideration of the entity link types traversed (1:1, 1:$n$ and $n$:$m$), despite the fact that, as Symons [235] acknowledges, they produce different processing requirements. The technique also counts a maximum of one reference to each entity per transaction, in spite of the fact that a transaction may refer to a given entity more than once. Mark II also fails to consider the types of operation that are performed in each transaction (that is, create, read, update or delete), even though others (British Gas [31]; Gray *et al.* [102]) suggest that the operations are of differing complexities. As justification, Symons [236] suggests that operation types should not be counted as they might depend on the logical database design, the file structure or the database tools used, that is, physical considerations. This, he suggests, is contrary to gaining a measure of the logical representation.

The use of McCabe's work as a basis for process complexity in terms of logical structure is certainly valid to an extent; however, evidence has also shown that McCabe's measure is not comprehensive enough to reflect overall complexity and that other contributors are assessed inadequately using this approach (Shepperd [221]). Therefore this basis should be further investigated. In calculating the input and output components, no distinction is made between data elements that are read

from/written to the database and those that are provided by/for the user, even though the processing and validation requirements for each of these situations may be quite different.

In order to perform estimation for future project requirements, historical effort data from past development projects must be allocated by staff after the fact to the input/output/process components and to each of the nineteen adjustment factors. Also acknowledged as crude in 1988, this method has appeared to provide reasonable results in validation studies. It is somewhat subjective, however, and may be jeopardised by leading questions from the assessor. Moreover, collection of the data required for the nineteen adjustment factors would be difficult to automate (King [148]). Finally, Albrecht [3] states that the use of local weights in the initial functional assessment makes the method invalid as a purely functional approach. This seems reasonable, in that he asserts that the functional measure should be derived first and then adjusted or weighted accordingly.

### Bang Metrics

Bang (DeMarco [68]) is offered as an implementation-independent, quickly derived approach for effort prediction that can lead to the development of size, cost and productivity estimates. The Bang system of measures is based on a three-view perspective of system specifications, ignoring all details of the method to be used in system implementation. The three views consist of a functional model, a retained data model and a state transition model. This complete representation enables the use of quantitative analysis to provide a measure of the function to be delivered by the system as perceived by the user. DeMarco [68] does state that most systems can be adequately specified using just two of the three views—particularly for business software this would normally consist of the data and functional models.

There are three main basic attributes that can be used as the principal indicators of Bang. They are the count of functional primitives or elementary processes (FP), the count of inter-object relationships (RE) and the count of data elements flowing out of the system (DEO). The ratio RE/FP is said to be a reasonable measure of data strength. If the ratio is less than 0.7, this implies a function-strong system—that is, a system that can be thought of almost completely in terms of operations, for example, robotic systems; if RE/FP is greater than 1.5, this implies a data-strong system, or one that should be thought of in terms of the data it acts upon. The middle range identifies hybrid systems. The DEO/FP ratio is indicative of the system's focus on either data movement or data computation. Commercial systems tend to have high levels of DEO/FP, scientific systems, low.

For function-strong systems it is suggested that the size or information content of a process can be approximated as a function of the number of tokens, or data elements, involved in the process. Variations in process complexity can then be accounted for through the assignment of weighting correction factors, based on sixteen functional classes, to the basic FP total. These weighted figures are then summed over all elementary processes to provide a final value of function Bang. The count of objects, or entities, in the database is the base metric for data-strong systems, with

some correction for the amount of connectedness among the objects. Data Bang is the overall result obtained by this procedure. Hybrid systems require separate computation of both function and data Bang so that the two figures can be used in the prediction of different activities. DeMarco [68] states that combining the two totals would be difficult, as it would be almost certain that one should be weighted more heavily than the other but that the magnitudes of these weightings would depend specifically on the system in question.

Consideration of complexity is therefore achieved in Bang through the use of weightings that are dependent on the flows of data elements or on the amount of entity connectedness. Although DeMarco [68] provides a beginning set of correction factors, these weightings must then be determined through trial and error and with extensive in-house calibration. The amount of work required by a department to determine the appropriate weightings has inhibited the wider use of Bang (Verner and Tate [252]). Furthermore, results for database-oriented systems, most common in the business domain, are sparse, despite the fact that the technique is now ten years old (Tate and Verner [243]).

Bang can be applied at the conceptual modelling phase and does consider the number of data elements processed. However, it fails to distinguish between input and output data elements, even though the effort required to develop their respective processing components is different (Symons [236]). Data Bang also considers the number of entity relationships, but no assessment of the relationship types is performed.

### Bang Metric Analysis

This is an adaptation of the original Bang method that considers both processing and data requirements in transaction-based systems (British Gas [31]). It is also primarily a project sizing technique. Each functional primitive or elementary process is assigned a level of complexity according to the number of create, read, update and delete operations that it performs, with each of these operations carrying a weighting factor. This forms the basis for the calculation of a process' function Bang. The formulation of data Bang is the same as in DeMarco's theory [68], that is, complexity is dependent on the number of entity relationships. Total Bang is the sum of both function and data Bang for each elementary process.

In terms of data-oriented transaction systems this is a much more useful approach, in that database operations are considered instead of DeMarco's sixteen weighted functional classes [68]. The weightings used for the operations were intuitively proposed, but have proved to be useful in validation. Regression techniques have been used to determine the appropriate coefficients for function and data Bang in the prediction of overall development effort. This method, however, still suffers from the same drawbacks as DeMarco's original proposal [68], that is, a failure to distinguish between input and output data elements and non-assessment of relationship types.

### Data Definitions

This is another approach that has been derived from DeMarco's Bang technique [68] (Fisher and Betteridge [88]). Not only are the number of functional primitives considered, but also such factors as the number of data definitions, database accesses, data flow lines and man/machine boundary crossings. The aim of the study was to investigate the effectiveness of these factors in the prediction of resource requirements. After an analysis of the relationships between these factors and effort data from one project, it was found that only the counts of data definitions and functional primitives were of significant influence. It was therefore decided to base the prediction on data definitions alone, as the number of functional primitives was deemed to be too dependent on individual decomposition strategies.

This approach also considers data and process at the same time, at a low level of analysis. However, the data consideration is superficial; it is not taken from the data structure itself but from the DFD process-based data dictionary. Thus no consideration of links between data elements or entities is performed.

### Usability Measures

Wilson [262] has described a method for determining the usability of systems, in order to enable the comparison of designs that conform to the same requirements. The approach is based on cognitive issues not generally covered in quantitative assessment. The procedure considers the number of user-visible concepts, terms and inter-relationships in a system, prior to implementation. This practice is said to actually measure the complexity of application problems, system designs and system-supported solutions, based on the semantic analysis of a design model similar to the ER representation. Under this model there are five mutually exclusive concept types:

1. entity – something that (usually) persists in time as (some of) its attributes and relationships change;

2. event – an occurrence of a change in the attributes and/or relationships of one or more things;

3. relationship – a directed association or connection between something and (usually) something else;

4. attribute – an aspect of something that can be qualitatively or quantitatively assessed;

5. value – an assessment of an attribute of something.

Different system design approaches, that is, using different methodologies, can be assessed for complexity using various factors, such as the number of entity types, the number of event types, the number of value types, the number of new terms and the average number of attributes per subject. Generally, the design method with the lowest total number of concepts and terms is the least complex and therefore

the most usable. Wilson suggests that the average values of the features mentioned should conform as a general rule to Miller's 7±2 constraint [178], which is believed to be related to understandability.

The complexity of solutions proposed for a system requirement can be measured using the following factors: number of entity types, number of entity attributes or relationships, number of event types, number of event attributes or relationships and the number of value types—these figures give the total concepts—and the average number of attributes/relationships per subject, the average number of events per subject, number of non 1 to 1 problem-solution choices (the number of times the user is faced with alternative ways to map problem concepts to solution concepts) and the number of non 1 to 1 problem-solution relationships (where a problem requires none or more than one solutions)—these values give the total number of problem-solution relationships. The solution with the fewest concepts is generally the one that supports the entities and operations with the best match to the problem and is therefore the easiest to implement. Again, Miller's constraint [178] is recommended for evaluation of the average figures.

Although a novel approach, this method has seen no further investigation. The focus on understandability reduces the usefulness of this technique as a general, objective procedure. The only consideration of processing in this scheme is the counting of entity event types and only the number of relationships is considered, not the type.

### Information Engineering Metrics

Data representing independent complexity variables thought to influence development phase effort was collected from a number of information engineering development projects (IE [123]). In producing an information strategy plan for an organisation it was found that the number of entity types had a large impact on project effort, based on twenty-eight projects from seventeen domains. Other important complexity variables were the number of lowest-level functions, the number of proposed data stores and several other factors relating to the structure and personnel of the organisation concerned. For business area analyses, the number of elementary processes to be implemented in a system was found to be highly influential, based on data derived from twenty projects over ten application domains. Other factors included the number of users interviewed, the number of relationships, the number of attributes and the number of action diagrams.

This approach is similar to the Fisher and Betteridge [88] study cited above, in that it is a practical, empirical evaluation of intuitive relationships with minimal background theory. The results obtained may be useful in the information engineering (IE) environment, but because the formulæ derived are totally oriented towards steps of the IE methodology, their general application may be less effective. Furthermore, the effort data was used after the fact for metric analysis. That is, it was not collected specifically for assessment purposes. Therefore much of the data was based on personal notes, personal memory, accounting data and best guesses. Finally, several variables relate to the development and organisational environment,

reducing the functional basis of the method. This may have been due to the fact that only some of the projects made use of CASE or similar tools.

### Entity Metrics

Gray *et al.* [102] describe a set of techniques for the assessment of the complexity of various tasks relating to the development of data-oriented systems. They firstly propose an ER metric for determining the effort required to implement a database design. There are said to be four factors that influence the complexity of a database design: the number of entities in the design, the number of relationships for each entity, the number of attributes for each entity and the distribution of relationships and attributes. The overall complexity of a complete ER diagram is shown as the sum of the complexities of the entities that comprise it. Individual entity complexity is calculated using the values of the number of relationships, functionally dependent attributes and non-functionally dependent attributes for each entity. Weightings for these factors are also used in the formula—it is suggested that these weightings can be used to reflect the impact of characteristics from the local development environment. The calculation also considers the 'functional complexity' of each entity, but this is assumed to have the constant value of one for every entity. A second metric proposed is the Area metric. This measure is derived from a Kiviat diagram representation of the same information used for the ER metric formulation.

The third measure is an enhancement of Shepperd's structural IF4 metric [223] which was itself derived from Henry and Kafura's Information Flow metric [118]. The original IF4 measure makes no consideration for the use of a database—therefore an extension is suggested. Each entity in a database is regarded as a type of module that can receive information, through create and update transactions, and can also provide information, through read and delete operations. A delete operation is said to be an information extraction because the entity will contain less information after the transaction is completed. Thus the enhanced IF4 metric (IF4+) is said to enable the assessment of both processing and data in a single metric approach.

Finally a measure of database operation complexity is proposed. This treats each operation (create, read, update and delete) as a virtual entity, being composed of the parts of the entities accessed by the operation. The ER and Area metrics as proposed can then be used, with the number of entities replacing the number of relationships in the original formula, to assess the overall complexity of each operation. Overall this would seem to be a very positive approach, particularly given that its focus is on data rather than on processing. Functions pertinent to the data are also considered, however, so processing is not completely ignored.

The decision to assign a delete operation as a provision of data is interesting. Although it is certainly true that the entity will contain fewer elements after the operation, it can equally be said that the operation itself is one that writes a blank record, therefore suggesting that it should be classified as a 'receive' by the entity. Placing this issue aside, the new IF4+ metric could be useful as a more comprehensive structural complexity measure. It is not strictly a functional measure, however, because the processing assessment is based on design-phase module structure charts.

The final measurement approach, considering database operation complexity, is also a valid and worthwhile proposal. Again, it would seem to be more comprehensive than many other techniques in that it attempts to consider processing and data in one metric. However, there is no indication as to whether one type of operation will be inherently more complex than another, without consideration of the data that it manipulates. Furthermore, the number and type of relationships between the entities are not considered, and there is no explicit guidance provided as to how entity look-ups or relationship exclusivity should be treated in the assessment.

### CAPO, CDM, SARA and STES Heuristics

Karimi and Konsynski [135] propose the use of a computer-aided tool to provide intelligent assistance in the development of code modules, based on the interaction of DFD processes. Computer-aided process organization, or CAPO, is said to be useful in the production of programs with highly cohesive modules with minimal coupling, low reference distribution and minimal transport volume. Yadav [267] suggests a similar approach in an attempt to assist software understanding and to contain ripple effect errors. The Control and Definition Modularization (CDM) method applies the theories of abstract data types and object oriented programming to traditional development with DFDs in order to produce more easily maintained module structures. Automated support has been provided to assist designers in this technique. Lor and Berry [166] describe a system architect apprentice (SARA) that uses DFDs and verification diagrams to produce system designs from semi-formal requirement representations. Tsai and Ridge [247] discuss the use of an expert system tool, the Specification-Transformation Expert System (STES), that assesses DFDs in terms of heuristics such as coupling, cohesion, fan-in and fan-out in order to produce a structural design of high quality. The values for the measures are determined by a combination of automatic and user-supplied data derivation, and are provided as feedback to encourage the analyst to iteratively refine the diagrams until 'suitable' measures are achieved—suitable measures are not defined, however.

Due to their sole basis in DFDs the underlying data structure is not considered by these approaches. More importantly for this discussion, the measures derived are oriented towards the quality of the subsequent design, rather than to the functional specification. Consideration of control, logic and timing dependencies, in CAPO at least, also means that some non-functional assessment is required, and the need for user input by some of the techniques may introduce an undesirable degree of subjectivity to the methods.

### MGM

The Metrics Guided Methodology (MGM) was proposed by Ramamoorthy *et al.* [201] as a reflection of the need for metrics from all development phases. Discussion of the specification stage is based on the use of requirements specification languages (RSLs). It is suggested that a spectrum of measures is needed to assess the different aspects of a specification, as it is normally not possible to specify requirements fully from just one perspective. Normally, then, both processing and data requirements

are developed. A set of metrics that considers the control-flow and entity models of an RSL specification is therefore described. Measures include the number of paths, nesting levels, ANDs and ORs, statements, data types and files.

Although this approach does consider the function of a system, the measurements used are more lexical or topological, due to the language-based form of RSLs. This also means that the technique is not applicable to conceptual data or structured analysis models.

### CASE Size Metrics

Tate and Verner ([242]; [243]) and Tate [240] assert that the automatic measurement of size as a function of data dictionary entries should be possible in a CASE environment. Furthermore, they state that the widespread use of graphics within CASE tools and the relative absence of lines of code means that more appropriate size measures should be chosen. They therefore suggest that measures of specification size applicable to transaction-oriented database systems may include those based on the data model, the data flow model and the user interface. Examples of specific measures suggested include counts of entities and attributes, data flows, processes and data stores. Complexity measurement, on the other hand, is described by Tate and Verner [242] as a relatively well-defined area of conventional development that should follow similar principles within CASE, except that it may be based on data structure and data flow models. At the risk of oversimplification, they suggest that complexity is a measure of component interconnectivity within a software product, an aspect that should be automatically computable within a CASE environment and that should present no particular problems.

Complexity in this study, however, is considered to be more than just connectivity. In fact, Tate and Verner's discussion of specification size [242] remains particularly appropriate here as size is certainly thought to have an impact on overall complexity. Therefore the measures suggested above are still relevant to this work. Their study is, however, a preliminary examination of possible metrics and consequently no evidence supporting or refuting their suggestions is provided.

## 3.5 Opportunities for Improvement

All of the approaches discussed above have some useful features and a few in particular would appear to be promising avenues for further research. Many problems, however, have also been identified. In particular, some of the approaches have been criticised for their lack of objectivity, in that much of the assessment can be directly dependent on decisions made by individual evaluators. This is in spite of the fact that automatic measurement extraction would now seem to be a prerequisite for any successful approach (Norman and Chen [191]). Some of the methods are not completely applicable at the conceptual modelling phase and some are also not comprehensive in their assessment. Most of the methods still suffer from a lack of significant validation and are therefore likely to remain underutilised in industry. Of those that have been tested, several have used correlation and regression analysis to

determine the desired relationships—these statistical methods, however, may have been inappropriate for the underlying data. Section 5.3.3 in Chapter 5 contains further discussion of this issue.

Another drawback of some of the techniques relates to the effect of environment dependence on the results obtained—a number of techniques stress that a significant amount of calibration is required, based on large pools of local historical data, if appropriate predictive information is to be derived. Moreover, much of the information obtained is only applicable at a system-wide level; that is, only a few of the techniques provide any guidance as to which parts of a system are likely to cause problems during subsequent development and enhancement. Finally, only a few of the measures have been proposed specifically for the purpose of complexity quantification. Consequently this characteristic may not have received the attention that it deserves, given its impact on the development process.

Clearly, then, there are a number of areas in which improvements to the assessment function could be made. Of particular importance are the issues of subjectivity, environment dependence, automatic collection and validation. All of these issues need to be addressed if any new method is to be accepted by the development industry. Any degree of subjectivity places too much emphasis on the working methods of particular individual assessors—if counting methods can be interpreted differently by individuals then the measures obtained from the same system by different people are likely to vary. Consequently any recommendations based on those measures will also vary. Any new method must therefore be totally objective to ensure consistent results and conclusions.

Similarly, independence from the influences of personnel, organisational aspects and the operating environment is also desirable if comparable and consistent results are to be obtained over time. Given the level of influence that automation has on the development process the impact of at least some of these characteristics is being significantly reduced. Proposed analysis approaches should therefore reflect this situation. As well as reducing the influence of subjectivity on the assessment procedure, automated data collection also lessens the work effort imposed on developers and assessors. Furthermore, automatic collection also reduces the risk of errors being introduced into the extracted data. Finally any new analysis procedure needs to be validated with real-world systems to illustrate that it is indeed effective in the relevant development domain. All of these issues are now addressed in the proposed analysis scheme, as discussed in the next chapter.

# Chapter 4

# Proposed Analysis Scheme

## 4.1 Introduction

The conclusions made in the previous chapter provide the basis for the development of a new complexity analysis scheme. A number of failings associated with previously suggested methods were described; the analysis scheme proposed here is a direct attempt to overcome several of these problems, as described in Table 4.1. In order to impose some degree of rigour onto the development of the scheme two semi-formal paradigms have been used; these are described in the following section, along with a discussion of the two-level assessment approach. This is followed by an examination of the various measures and of the reasons and assumptions upon which the selection of measures was based. A short summary of the proposal then concludes the chapter.

## 4.2 Scheme Development

Several proposed complexity measurement approaches have been extensively criticised for being inapplicable until late in the development process (Samson *et al.* [217]; Londeix [164]). Given that the correspondence between specifications and final systems is very close in 4GL- and CASE-developed software (Harel [111]; see Figure 3.1), functional complexity indicators should prove to be useful in the early discrimination of specification structures and in assessing the impact that these structures have on other process and product attributes. Moreover, concentration on complexity, rather than just size, enables the consideration of more than just one dimension of software. In order to determine the specific measures that might be useful in assessing the complexity of specifications, the Goal/Question/Metric (GQM) paradigm, as developed by Basili and others (Basili and Rombach [10]; Basili and Weiss [11]) and enhanced by Shepperd [223], and Bush and Fenton's Classification Scheme [37] have been adapted and used in this study. These approaches encourage the structured selection of appropriate measures, using a process of decomposition and partitioning from high-level goals, until the data elements required to achieve the goals have been specified. The application of the two procedures to the goals of this study is shown in Figures 4.1 and 4.2. Due to the rather cluttered nature of

Figure 4.1, further explanation of the figure's structure and of the metrics chosen is provided here and in the following section.

| Problem | Solution |
|---|---|
| Subjectivity | The scheme is totally based on the functional specification of system requirements; consequently, all of the measures are directly quantifiable in an unambiguous, assessor-independent manner. |
| Manual Collection | As all of the measures are derived from requirements models that are widely used in automated development tools, collection of the measures can be easily incorporated into these tools so that collection errors can be avoided. |
| Late Derivation | The requirements specification is one of the earliest available products of the development process, enabling rapid measurement determination. |
| Narrow Focus | Since a specification can be considered from a number of perspectives, for example, data, process and/or user interface, measures applicable to each perspective have been included in the scheme. |
| Environment Dependence | Given that automation plays a significant part in the development of systems when CASE tools and 4GLs are fully utilised, it is asserted that the development environment will have far less impact on the data obtained from different sites; therefore results from different environments may be more easily compared. |
| Need for Calibration | As a result of the last point it is also suggested that a lesser degree of calibration will be needed, enabling more rapid uptake of the analysis recommendations by organisations that do not have pools of recent project data. |
| System-wide Results | The scheme is two-tiered so that results are available at both the elementary function and system level; this provides a greater degree of accuracy and flexibility to the project manager. |
| Various Goals | This scheme is specifically intended to be a complexity analysis technique and is therefore centred on assessing those features that are thought to contribute to complexity. |
| Lack of Validation | The scheme is to be validated with actual systems developed within the commercial software industry at more than ten different sites. |
| Poor Statistical Analysis | The underlying data distributions that are derived from the analysis procedure will be thoroughly examined so that appropriate statistical techniques can be determined; this will reduce the risks of result misinterpretation. |

Table 4.1: Previous failings and current solutions

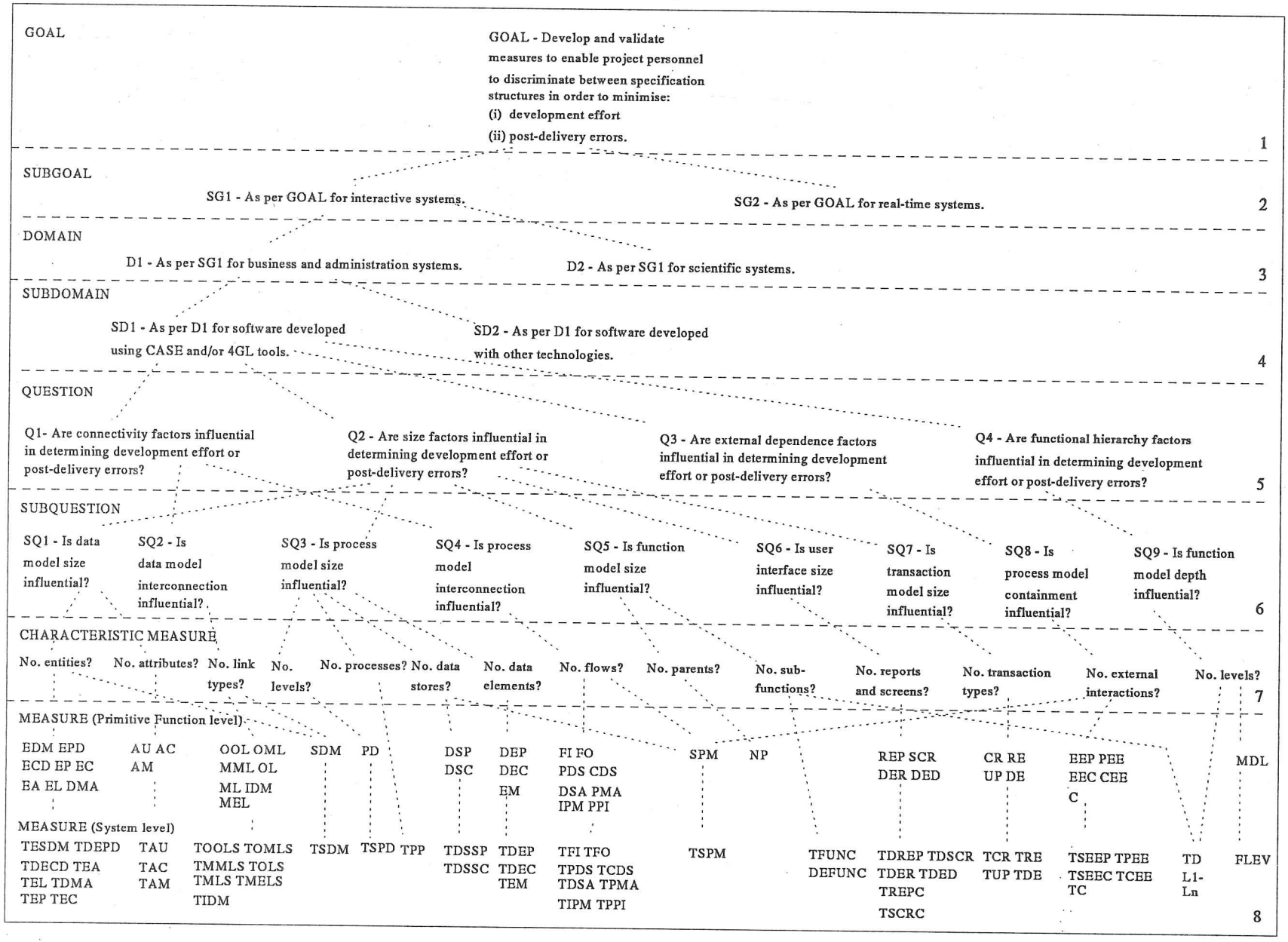



Figure 4.1: Goal/Question/Measure paradigm

The overall goal of this study is to develop and validate measures that enable development personnel to discriminate between specification structures at both the system and primitive function level in order to determine development effort and post-delivery error occurrence. This is the goal as specified on Figure 4.1 at level one, the level being indicated by the numbers down the right-hand side of the figure. The subgoals, domains and subdomains of the paradigm represent a refinement of the area to which the study, and therefore any result, applies. It can therefore be seen from the information depicted at levels two, three and four of the figure that this study is applicable to interactive business and administration systems developed with CASE tools and/or 4GLs. Questions about the influence of possible determinants of effort and errors are then added to the hierarchy at the next level, level five. These are subsequently broken down into subquestions at level six so that the questions may be applied to distinct parts of specifications for the type of system described above.

General characteristic measures are then derived at level seven, followed by the list of specific measures that are to be recorded in this study. Abbreviated forms of the measures are used at level eight on the figure in an attempt to aid in its clarity. The complete lists of measures along with their formal definitions appear in the next section in Tables 4.2 to 4.13. Again to reduce cluttering only the specification measures are given on the figure and in the tables—the project management measures that are to be used to assess the analysis scheme are instead described in Figure 4.2. The final selection of measures, as depicted at level eight in Figure 4.1, has been broken up into two sections, primitive function measures and system measures. Separate diagrams could have been used to illustrate the development of the two sets of measures; however as this would have necessitated repeating seven of the eight diagram levels it was thought that one diagram would be sufficient. This two-tier assessment approach reflects the need for both system-level (macroanalysis) and primitive function-level (microanalysis) analysis of complexity for resource allocation and progress tracking.

The classification illustrated in Figure 4.2 provides essentially the same outcome as Figure 4.1, but it also includes more explicit details of the particular assessment and estimation goals of the current study. Bush and Fenton [37] suggest that entities in the software engineering domain may be classified as processes, products or resources. In this study resource information, such as hardware usage, is of no direct interest. Thus only the product and process classes are considered here. The partitioning of the classes continues until mutually exclusive and exhaustive classes of measures have been determined. Thus in Figure 4.2 the effort and number of occurrences (No. Occ.) information for the process class is eventually partitioned into specific indicators of effort for the development phases and into error indicators for the acceptance testing phase. Similarly for the product class, product complexity of a functional specification is partitioned into four components—size, connectivity, external dependence and hierarchy—and these components are in turn decomposed until the number of occurrences of mutually exclusive product classes are specified.

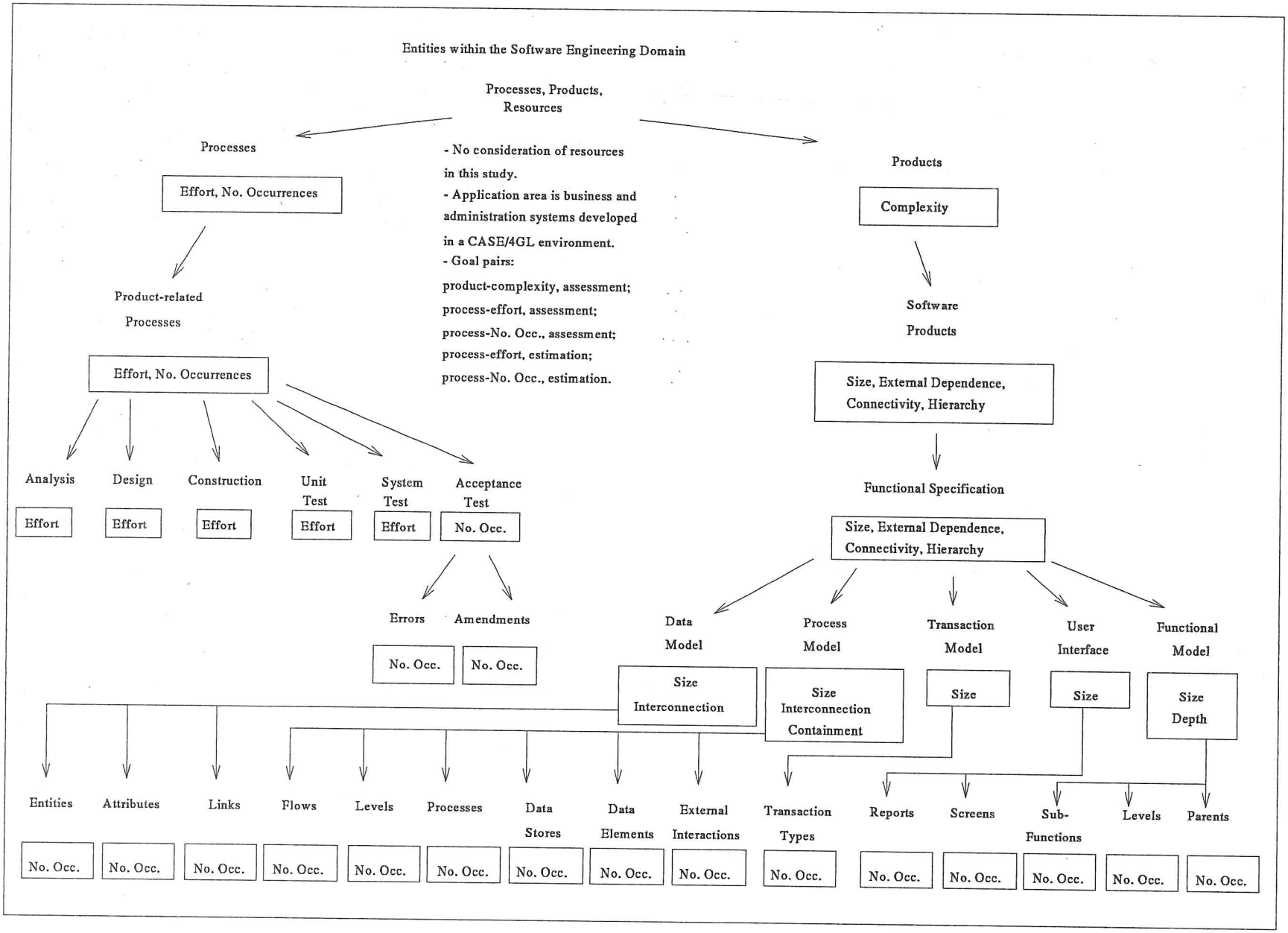



Figure 4.2: Classification paradigm

The two figures, 4.1 and 4.2, therefore illustrate all of the data elements to be collected for the validation of the scheme. The Goal/Question/Measure figure lists all of the functional product measures required to achieve the goals of the experiment, but for reasons of clarity it does not include the project management data required. Conversely the classification figure breaks the project management data down into simple data elements, but again for reasons of clarity it only specifies the characteristic product measures, which correspond to those at level 7 of Figure 4.1, rather than the lower-level measures. Both figures, however, provide the basis for the data collected in this study—the measures are therefore fully defined in the next section and in the following chapter.

In some cases one of the two function-oriented specification methods considered in the scheme, that is, data flow diagrams and functional decomposition hierarchies, may not be used in a given development project. Data collection will therefore follow one of these three procedures:

1. if both DFDs and FDHs are used then each elementary DFD process should map to a corresponding low-level FDH module—in this case, all five sets of measures should be taken; that is, process model and functional model measures, in association with the transaction, user interface and data model measures;

2. if only the DFD is broken down to an elementary level then process model measures only should be recorded, in association with the transaction, user interface and data model measures;

3. if only the FDH is broken down to an elementary level then functional model measures only should be recorded, in association with the transaction, user interface and data model measures.

Thus for the purposes of this study a *primitive function* consists of:

- a single function at the lowest level of a hierarchically decomposed functional model (referred to from now on as a *functional model primitive*) **AND/OR**

- a single process (and all connected elements) at the lowest level of a hierarchically decomposed process model (referred to from now on as a *process model primitive*) **AND**

- the section of the data model upon which the function and/or process acts (referred to from now on as a *data model primitive*).

This approach may be illustrated by a small example taken from the specification of a university department's administration system. In this case, only DFDs have been used to depict the process requirements of the system. At the elementary level there is a process that specifies the production of a class list. Given that Figure 4.3 depicts the underlying data model for the whole system, Figures 4.4 and 4.5 show the relevant process and data model primitives that comprise the single primitive function.

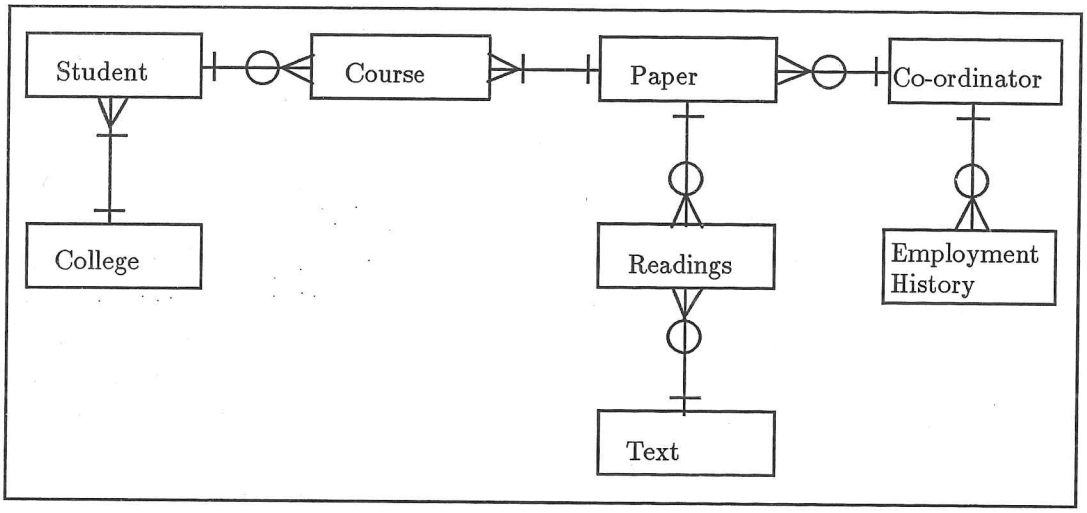



Figure 4.3: University department system data model example

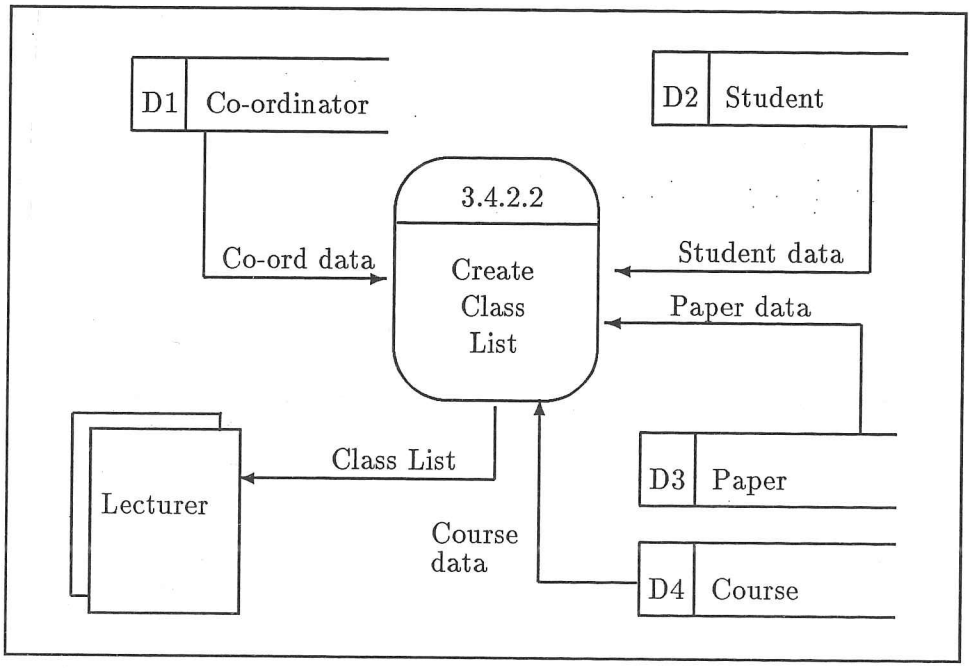



Figure 4.4: Process model primitive example

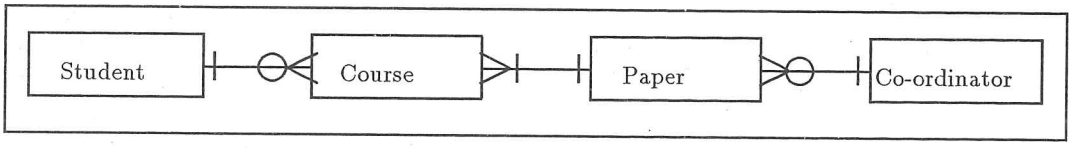



Figure 4.5: Data model primitive example

## 4.3 Rationale and Expectations

Although a foundation for the measures has been established, through the investigation of commercial software specification methods and by utilising the GQM paradigm and Bush and Fenton's classification scheme [37], the reasons for the inclusion of some of the measures may be unclear. The next six sub-sections (4.3.1 to 4.3.6) therefore describe the rationale behind the measurement selection procedure by considering the expected impact of specification characteristics on the software development process. Justification of the general approach is followed by a discussion of the measures derived from each of the five specification perspectives. Small examples are used to illustrate the determination of values for each set of measures.

### 4.3.1 General Approach

DeMarco [68] suggests that the effort required to implement a system increases monotonically with increasing specification size, assuming that there is no redundancy in the specification. As CASE helps to ensure that redundancy is kept to a minimum, size measures derived from specification models should prove to be useful in effort determination. DeMarco [68] also claims that the size of a specification model approaches invariance with respect to the decisions of the individual modeller. This would suggest that, for a given system requirement, roughly the same functional measurement values will be obtained irrespective of the modeller, particularly when CASE technology is employed (Robinson [204]; Tate and Verner [242]). This does not imply that the same specification will be produced, but only that the scope and size of the specifications will be similar. However this in itself is an aid to achieving more consistent assessment and analysis. Moreover, different specifications will provide different measures that will in turn result in different effort estimates. Thus the basis of the complexity analysis scheme in specification representations would still appear to be sound.

A similar comment to that of DeMarco's above is made by Rudolph [213] in a discussion on functional size assessment—the remark is equally applicable to any type of functional analysis, however. Rudolph suggests that by its very nature functional assessment should not in the first instance reflect 'external' factors such as programmer and organisational experience or the effects of various implementation approaches. Rather, the absolute value of the measured attribute should be established first, and then adjusted if required. This is the 'fault' cited by Albrecht [3] concerning Symons' Mark II FPA method [236]. Furthermore, Kitchenham [151] provides empirical evidence for the assertion that programmer ability, team size and personnel experience indicators have no influence on effort requirements in a single environment. Although the systems that are to be used for validation in this study have been collected from a number of sites it may be reasonable to assume that they will comprise a similar environment, in that the systems are all of one type—transaction processing and reporting—and they were all developed using automated tools. Moreover, Chen and Norman [43] suggest that the use of a graphic interface

in many of the tools enables easier learning and use, lessening the impact of tool experience on effort requirements. All of these factors lend support to the approach adopted in this study, that is, the assessment of functional complexity with no initial adjustment for other factors.

Further support for the purely functional approach is indirectly provided by the advantage of application portability that many CASE tools now provide (Banker and Kauffman [9]). Tate and Verner [243] suggest that as projects move through the phases of development more specific methods and techniques are employed. They go on to suggest that this leads to increasing dependence of software products on the target technology, even in a CASE environment. The implication is that the development and implementation methods chosen have an increasing impact on a system's size and that this will have a corresponding impact on development effort. However several CASE tools enable development to be performed irrespective of the eventual implementation platform, as the tools include facilities for implementation on a number of different platforms (Brown *et al.* [34]; King and Warren [149]; Banker and Kauffman [9]). Thus functional measures should provide a sound basis for the determination and classification of effort requirements in an automated environment up until the implementation phase of development.

In the past, adjustment of functional measures has generally been performed to enable the consideration of special system requirements. It has recently been suggested, however, that systems developed with 4GLs and CASE tools will not require adjustment as the 'special' requirements will be developed as standard (Symons [235]; Symons [236]; Tate and Verner [242]; Chen and Norman [43]):

> In the longer term, the ultimate computer-aided systems engineering (CASE) tool will provide all these technical features automatically; we shall only have to think about the information-processing requirements of the problem. In this ultimate situation, the coefficient [of the adjustment factor] will fall to zero, and there will be no further need for a Technical Complexity Adjustment (Symons [236], p 29).

Although it is questionable as to whether we have reached the age of the 'ultimate' CASE tool, this assumption is the basis for the proposals being tested here. That is, that useful assessment can be performed based solely on systems' functional requirements. As 4GLs and CASE tools tend to specify functional transactions rather than procedural components, Verner *et al.* [254] suggest that a specification produced with these tools will be a closer representation of the pure inherent function required. Moreover, since development using these tools is often based around a central repository, the likelihood of duplicated and inconsistent work among development teams should be reduced, providing a basis for more consistent and accurate assessment of functional characteristics. It is certainly possible that measures from a specification will be related to the functional value of the system, based on the assertion that transactions comprised of larger and more complex specifications provide more function to the user (Verner *et al.* [254]). Appropriate measures of functional size and complexity are therefore required. Those proposed here are a first response to this need.

A major criticism of many existing measurement methods is their assessment of only one aspect of complexity, for example, control-flow, size or data flow (Case [40]; Weyuker [260]; Longworth *et al.* [165]; Jayaprakash *et al.* [131]). The proposed analysis scheme of this study should at least partially overcome this problem, in that size, interconnection, data flow, data structure, process coupling and overall function are all assessed in some way. The comprehensive approach that has been adopted in this proposal is supported by Tate and Verner [242] and Wrigley and Dexter [265]; they suggest that appropriate specification measures should come from data structure and data flow models and from aspects of the proposed user interface. Each of these representations has been considered in the overall analysis scheme. The general expectation underlying this approach is that systems or primitive functions that return high values for the transaction, functional model, user interface, process model and/or data model measures will be more time-consuming and error-prone to develop than systems/primitive functions that result in low measures. Due to its more comprehensive approach, the scheme also includes measures that are applicable to software specification products not normally considered. Grady's examination of software development work-product analysis states that the design stage "...includes work products for prototypes and data dictionaries because they are widely used today, although they represent two cases where metrics research has been very limited (and so no metrics are shown)" (Grady [101], p 30). In the proposed scheme the data dictionary may be partially assessed by the data model and process model measures, and the prototype by the user interface and functional model measures (Clarke [51]).

The two-level approach to assessment is another important aspect of the overall analysis scheme. Although system-wide measures and indicators are useful, it has been acknowledged that lower-level analysis is necessary for effective project management. Verner *et al.* [254] and Stevens [233] remark that elementary function-based assessment is required so that resource allocation in subsequent development phases can be carried out more effectively. Moreover, in terms of accuracy, lower-level assessment receives further support—Stevens [233] and IE [123] suggest that a more detailed functional breakdown will provide more accurate measures, due to the reduced variance achieved.

Furthermore, the measures in this proposal are automatically derivable at a very early stage in the development process, increasing the scope for objective and effective estimation and discrimination. According to Grady [101] and Gray *et al.* [102], one of the most promising aspects of CASE is the facility for automatic, 'on-line' delivery of metrics to the project manager. The notion of automatic metric collection and analysis has widespread support and would appear to be essential in an unobtrusive form if metric analysis is to become a useful, integral part of the development process (Henry and Lewis [119]; Norman and Chen [191]). Automatic collection would also enable more effective progress reports to be produced—as long as subsequent functional changes to a system were incorporated into the current specification models, revised measures and/or estimates could be automatically generated for the project manager during all subsequent stages of development.

Thus specification-based functional complexity analysis as a general approach

would appear to have extensive support in recent literature. The purpose of the following five sections, however, is to fill out the details of the specific scheme adopted in this study. A number of commercial software specification techniques were described in Chapter 2—methods for the assessment of functional representations developed using these techniques are therefore now provided. Each of the sections describes the basis for assessment and the actual measures chosen in the current scheme. Measures preceded by an asterisk (*) in Tables 4.2 to 4.13 are composite measures; that is, they are merely calculated from the values of other measures. The use of composite measures as overall indicators of specification perspective size is supported by Tate and Verner [243]. This approach is therefore extended in this study to consider aspects other than size, for example, interconnection.

In cases where a non-composite system level measure is simply the sum of the values obtained for the same primitive function level measure over all primitive functions in the system it is denoted by placing a 'T' in front of the primitive function measure. Thus the 'TCR' measure in Table 4.3 is the sum of the values of the 'CR' measure that are obtained for all primitive functions in the system. For example, if a system was made up of four primitive functions and the values of the CR measure for those primitive functions were two, one, three and zero respectively, then the value of the TCR measure for that system would be the sum of these four figures, that is, six. (A small example such as this is provided in each of the following five sections to illustrate the derivation procedures for measurement values.)

### 4.3.2 Transaction Measures

Transaction details are commonly specified for database manipulation systems in the commercial environment. Low level transactions in these systems always perform at least one of the following operations: create a record (C), read a record or part of a record (including look-up validation) (R), update a record or part of a record (U) or delete a record (D) (CRUD). Although generally recognised as a data structure-based specification method, due to its basis in entity models, Kerr [137] in fact describes process modelling solely in these terms, in that for each entity in a data model at least three functional modules will be developed, with each module containing the create, update or delete rules for controlling the manipulation of the data. Gray *et al.* [102] remark that single create, update and delete operations work on only one entity each, but that a read may access several entities in one operation. They therefore suggest that the read operation may be different from the others in terms of complexity. Worsley [264] also found that in a study of enhancement effort for a medium-sized 4GL system, estimates for tasks that required new create transactions were poor, and that the actual re-testing effort required for these enhancements was also generally greater than predicted. Furthermore, the four operations are also weighted differently under a project sizing methodology developed by British Gas [31].

| Microanalysis transaction measures | |
|---|---|
| CR | Number of create transactions performed by the primitive function |
| RE | Number of read transactions performed by the primitive function |
| UP | Number of update transactions performed by the primitive function |
| DE | Number of delete transactions performed by the primitive function |

Table 4.2: Primitive function level transaction measures

| Macroanalysis transaction measures | |
|---|---|
| TCR | Total number of create transactions performed by the system |
| TRE | Total number of read transactions performed by the system |
| TUP | Total number of update transactions performed by the system |
| TDE | Total number of delete transactions performed by the system |

Table 4.3: System level transaction measures

Four size measures relating to the transaction representation of a specification are therefore included in the analysis scheme, at each of the primitive function and system levels. These are the (T)CR, (T)RE, (T)UP and (T)DE measures, as defined in Tables 4.2 and 4.3. Note that it is the number of operations, not the number of entities operated upon, that is counted in the scheme; the number of entities referenced is assessed in the data model measures. Thus if a single primitive function reads and updates a single entity, both the RE and UP measures for that primitive function should be incremented.

The transaction measures may be illustrated by the following example. A system comprises three primitive functions, F1, F2 and F3. Function F1 reads data from two entities for subsequent processing by another primitive function; F2 reads data from one entity, displays it on the screen and then updates that entity after input from the user; F3 deletes records from two entities. The transaction measures for these primitive functions therefore take the values shown in Table 4.4. The system level transaction measures are then easily determined from the primitive function values, using the method described at the end of the previous section. Thus for this same example TCR equals zero, TRE equals three, TUP equals one and TDE equals two.

| Measure | F1 | F2 | F3 |
|---|---|---|---|
| CR | 0 | 0 | 0 |
| RE | 2 | 1 | 0 |
| UP | 0 | 1 | 0 |
| DE | 0 | 0 | 2 |

Table 4.4: Example transaction measures

### 4.3.3 Functional Model Measures

Paulson and Wand [196] suggest that functional decompositions are central to most development approaches. In cases where the functional model is broken down to a level at which elementary modules are depicted, this representation can provide a quantitative insight into several aspects of the specified system. Primarily, system size, in terms of the number and distribution of modules, is shown. However the calling structure is also portrayed, through the use of linkages between levels of the hierarchy. Measures of both system size and system depth are therefore available from this representation. These measures are defined in Tables 4.5 and 4.6.

| Microanalysis functional model measures | |
|---|---|
| MDL | Maximum decomposition level of the function |
| NP | Number of parent functions |

Table 4.5: Primitive function level functional model measures

| Macroanalysis functional model measures | |
|---|---|
| DEFUNC | Number of distinct elementary functions in the decomposition |
| FLEV | Maximum number of function decomposition levels |
| L1 | Number of functions at level 1 |
| L2 | Number of functions at level 2 |
| ⋮ | ⋮ |
| L$n$ | Number of functions at level $n$ |
| *TFUNC | Total Functions (L1 + L2 +...+ L$n$) |
| *TD | Total Decomposition ((L1 × 1) + (L2 × 2) +...+ (L$n$ × $n$)) |

Table 4.6: System level functional model measures

Only two functional model measures are included in the analysis scheme at the primitive function level. The first measure, MDL, is an indication of the level of system depth at which the function is to be implemented. Modules in a hierarchy, and the data elements that they manipulate, are likely to be affected by the processing performed above and below them. Those at higher levels, for example, may be more vulnerable to functional errors, due to the fact that they must control and co-ordinate the processing of often large numbers of modules at lower levels of the hierarchy. Greater caution in development and more extensive testing may therefore be necessary for higher level modules—it is suggested here that the MDL measure may to some degree reflect these requirements. The second measure, NP, quantifies the number of parents that call a single function. By examining the use of common function calls it may be found, for example, that a function that is called or controlled by more than one parent is more difficult to develop than a module with only one parent, given that the plural-parent module may need to cope with different data under different calling conditions.

A completely different set of measures than that used for primitive function level assessment are included in the analysis scheme at the system level. They are mainly indicators of system size, but there is also some consideration of the impact of the decomposition structure on overall complexity. DEFUNC is the count of all the distinct elementary system functions, that is, those functions in the hierarchy that call no other functions. Note that each distinct function is only counted once, even though it may be called in several instances. This is to reflect the fact that the function will only be developed once, even if it may be used more than once. The FLEV measure is similar in principle to the primitive function level MDL measure. If a particular system hierarchy is decomposed down to a maximum of six levels, that is, one or more elementary functions must be traced back through at most six calling modules to reach the highest-level system description, then FLEV will equal six for that system. Comparative indications of system depth may contribute in a relative manner to overall system complexity. The L1 to L$n$ measures are derived only for the determination of the TFUNC and TD measures of decomposition. Note that level 1 is one below that at which the highest-level system description is depicted. The TD measure uses relative weightings to provide an overall assessment of the complete decomposition structure. This is in an attempt to assess the relative contributions of both hierarchy depth and breadth to total system complexity.

The hierarchy depicted in Figure 4.6 provides a basis for illustrating the derivation of the measures just described. If we choose function 'Store Item' for micro-analysis assessment the MDL measure would take a value of two; this is because function 'Store Item' occurs at level 2 in the hierarchy. The NP measure for this same function would equal one as it has just the one parent function, that is, function 'Receive and Store Item'. If, on the other hand, we were to choose function 'Read Item and Required' for assessment, MDL would be assigned the value of five, as this is the highest value level at which it appears. Furthermore NP would equal two for this function, as it is called by both function 'Read Updated Items' and function 'Calculate Deficit'. Any repeat of a function such as in this case is denoted on the diagram by an asterisk (*) after the function name.

Concentration on the complete hierarchy also enables the derivation of the macro-analysis functional model measures. The DEFUNC measure is defined as the number of distinct elementary functions in the decomposition. For this example the distinct elementary functions, that is, those that do not call any other functions, are functions 'Receive Item', 'Store Item', 'Read Item and Required', 'Request Missing Items' and 'Report Requirements'. Hence DEFUNC for this example equals five. The L1 to L$n$ measures are as follows: L1 equals two (functions 'Receive and Store Item' and 'Determine Requirements'), L2 equals four (functions 'Receive Item', 'Store Item', 'Identify Missing Items' and 'Request Missing Items'), L3 equals three (functions 'Read Updated Items', 'Request Missing Items' and 'Report Requirements'), L4 equals two (functions 'Read Item and Required' and 'Calculate Deficit') and L5 equals one (function 'Read Item and Required'). The FLEV measure is equal to $n$ in the L$n$ measure, that is, FLEV equals five for this example. The TFUNC and

TD measures are then directly computable from the previously derived measurement values.

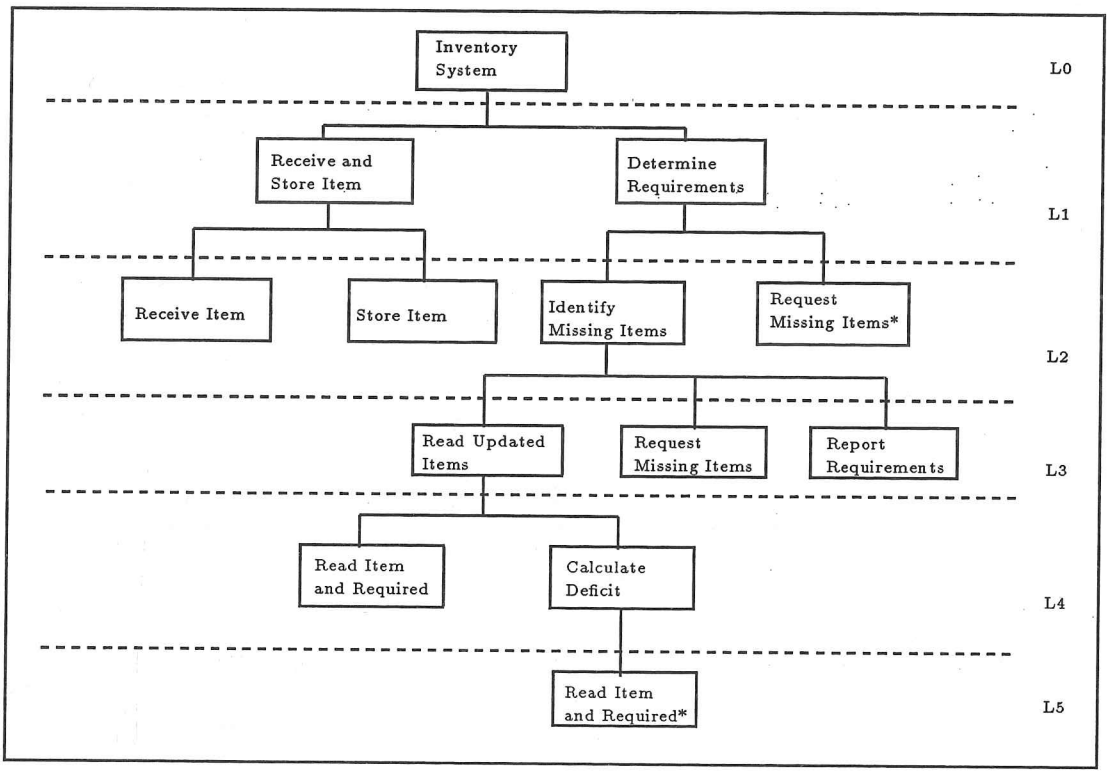



Figure 4.6: Functional decomposition hierarchy example

### 4.3.4 User Interface Measures

Particularly for software development in a 4GL environment the number of screens, reports and data elements produced for the user is expected to have a significant impact on development effort, as the creation of acceptable prototype screen and report formats is often a major part of 4GL system production (Boehm *et al.* [26]; Lin [161]). Worsley [264], for example, found that the time taken for report development with a 4GL was longer for reports that included complex layouts and that accessed large numbers of tables.

| Microanalysis user interface measures | |
|---|---|
| REP | Number of reports produced by the primitive function |
| DER | Number of distinct data elements reported by the primitive function |
| SCR | Number of screens displayed by the primitive function |
| DED | Number of distinct data elements displayed by the primitive function |

Table 4.7: Primitive function level user interface measures

| Macroanalysis user interface measures | |
|---|---|
| TDREP | Total number of distinct reports produced by the system |
| TREPC | Total number of report calls performed by the system |
| TDER | Total number of distinct data elements reported by the system |
| TDSCR | Total number of distinct screens displayed by the system |
| TSCRC | Total number of screen calls performed by the system |
| TDED | Total number of distinct data elements displayed by the system |

Table 4.8: System level user interface measures

The primitive function level user interface measures are therefore included to reflect findings such as this—the layout of both reports and screens is considered here to be related to the number of data elements that are produced on each. It is also assumed that a primitive function that uses more screens and produces more reports will be more complex than one that uses fewer of these representations. The REP and SCR measures, as defined in Table 4.7, therefore consider the number of complete reports and screens that are referred to by a primitive function, and the DER and DED measures consider the number of distinct elements used on those reports and screens. A distinct element is an actual data element, not a label, header or footer, that should be counted only once for each report or screen on which it appears, no matter how many times that element may be used on a single report or screen. The system level measures for this representation, which appear in Table 4.8, are similar to but not the same as the primitive level measures just described. The TDREP and TDSCR measures are simply the total number of distinct, that is, different, reports and screens that are employed by a system. TDER and TDED are directly comparable to the DER and DED measures discussed above; thus they are simply the sums of the DER and DED values for all primitive functions. The two other indicators, TREPC and TSCRC, equate to the total number of times that reports and screens are used in the system.

The sample screen shown in Figure 4.7 may be associated with a given primitive function. If this is the only screen used by the primitive function, and no reports are produced by the function, then the microanalysis measures would take the following values: REP and DER would both equal zero, SCR would equal one and DED would equal thirteen. The thirteen elements displayed are Order No., Date, Date Filled, Cust. Ref., Back-O?, Component, Required, Available, Back-O, Comp. Cost, Line Cost, Sub-Total and Total.

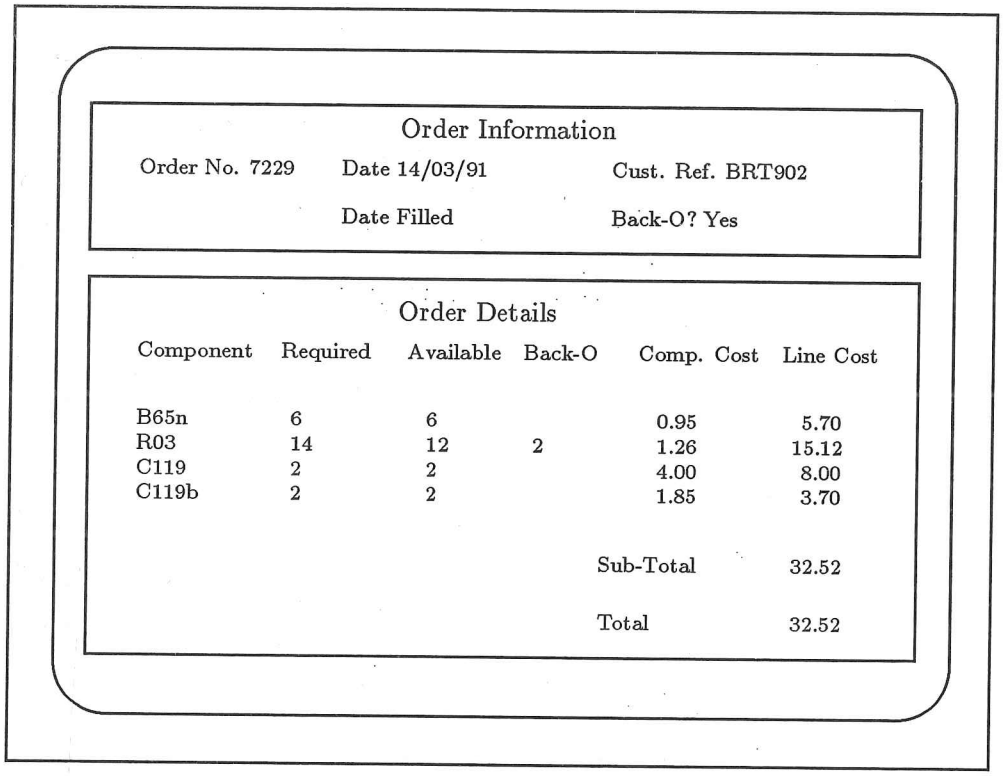



Figure 4.7: Screen example

### 4.3.5 Process Model Measures

Keuffel [143] states that the underlying objective of the use of DFDs is to partition systems in order to reduce complexity. Therefore an assessment of DFD representations should provide useful indications of relative complexity levels. Generally the measures of this model from both analysis levels reflect the assumption that a large process model, in terms of decomposition levels, processes, related data stores and individual data element usage, will result in a proportionally large coded function. Thus the measures from Tables 4.9 and 4.10 that relate to the numbers of levels, processes, stores and elements, that is, TPP, PD, TSPD, DSP, DSC, TDSSP, TDSSC, (T)DEP and (T)DEC, are all indications of process model size. The TPP measure is clearly only applicable to the system-level analysis procedure, given that the micro-analysis technique considers only one primitive process at a time. Therefore TPP is simply the total number of primitive (lowest-level) processes in the system. In this respect it is very similar to the DEFUNC functional model measure. In the same way, the process model PD and TSPD measures are similar to the functional model MDL and FLEV measures. They are included to assess the depth of processing that is to be implemented in the final system.

| Microanalysis process model measures | |
|---|---|
| FI | Number of flows into the process |
| FO | Number of flows out of the process |
| DSP | Number of distinct data stores providing data |
| PDS | Number of provisions from data stores |
| DSC | Number of distinct data stores consuming data |
| CDS | Number of consumptions by data stores |
| DEP | Number of non-file data elements produced by the process |
| DEC | Number of non-file data elements consumed by the process |
| PPI | Number of process-to-process flows into the process |
| PD | Process depth (Number of parent processes up to level 1) |
| EEP | Number of distinct external entities providing data |
| PEE | Number of provisions from external entities |
| EEC | Number of distinct external entities consuming data |
| CEE | Number of consumptions by external entities |
| *PMA | Process Model Access (FI+FO) |
| *IPM | Interconnection (Process Model) $(\text{FI} \times \text{FO})^2$ |
| *DSA | Data Store Access (PDS+CDS) |
| *EM | Element Manipulation (DEP+DEC) |
| *C | Containment (PEE+CEE) |
| *SPM | Size (Process Model) (PMA+DSP+DSC+EEP+EEC) |

Table 4.9: Primitive function level process model measures

It is acknowledged that the data store measures DSP, DSC, TDSSP and TDSSC may or may not be of significance, as the number of stores may be determined by an arbitrary decision of the analyst. One analyst may prefer to have a distinct data store for each entity, whereas another may group entity-views into data stores simply for representational convenience. For purposes of completeness, however, these measures will still be taken for each system. The use of the words 'non-file' in the (T)DEP and (T)DEC definitions reflects the fact that all file-related elements are assessed in the evaluation of the data model with the (T)AU and (T)AC measures. There could be considerable overlap in these two pairs of measures; the 'non-file' condition, however, enables the sole consideration of data elements other than those stored by the system that are (i) input by the user or by other external systems/processes and (ii) produced on the screen and in report formats.

Process model interconnection should reflect the degree of coupling that will be implemented in the final system (Tsai and Ridge [247]). Interconnection at this level may be related to Henry and Kafura's design phase Information Flow measure [118], in that the fan-in and fan-out measures may be approximated by the flows-in and flows-out measures of this proposal. Since it is generally accepted that DFD process interconnection should be minimised to lessen complexity (DeMarco [68]; Hawryszkiewycz [117]; Tan *et al.* [238]), lower values of interconnection measures such as (T)FI and (T)FO should have a positive impact on the ease of development.

| Macroanalysis process model measures | |
|---|---|
| TPP | Total number of primitive processes |
| TFI | Total number of flows into primitive system processes |
| TFO | Total number of flows out of primitive system processes |
| TDSSP | Total number of distinct system data stores providing data |
| TPDS | Total number of provisions from data stores |
| TDSSC | Total number of distinct system data stores consuming data |
| TCDS | Total number of consumptions by data stores |
| TDEP | Total number of non-file data elements produced by the system |
| TDEC | Total number of non-file data elements consumed by the system |
| TPPI | Total number of process-to-process flows into system processes |
| TSPD | Total system process depth (Number of process levels) |
| TSEEP | Total number of distinct system external entities providing data |
| TPEE | Total number of provisions from external entities |
| TSEEC | Total number of distinct system external entities consuming data |
| TCEE | Total number of consumptions by external entities |
| *TPMA | Total Process Model Access (TFI+TFO) |
| *TIPM | Total Interconnection (Process Model) $(\text{TFI} \times \text{TFO})^2$ |
| *TDSA | Total Data Store Access (TPDS+TCDS) |
| *TEM | Total Element Manipulation (TDEP+TDEC) |
| *TC | Total Containment (TPEE+TCEE) |
| *TSPM | Total Size (Process Model) (TPMA+TDSP+TDSC+TEEP+TEEC) |

Table 4.10: System level process model measures

The level of system containment is also considered to be important. Dependence on data supplied from outside the system boundaries may have an effect on the ease of system implementation and maintenance, especially where control over the form and validity of the data is out of the developer's hands. External entities that receive data, on the other hand, often require this information in some form of report. For example, a packing slip may be sent to a warehouse, or an invoice to a customer. Therefore development and maintenance of the relevant process would also involve the creation or consideration of a report form and the incorporation of extra processing to produce that report. The EEP, EEC, TSEEP, TSEEC, (T)PEE and (T)CEE measures are therefore included to reflect the impact of process model containment. The measures differ in the fact that the first four simply consider the number of external entities that are involved in the operation of a primitive function or a system, whereas the final four assess the actual number of interactions between system processes and external entities. This follows the same approach as that used in the collection of the DSP, DSC, TDSSP, TDSSC, (T)PDS and (T)CDS measures.

The process model primitive example shown earlier in the chapter in Figure 4.4 may be useful in illustrating the derivation of microanalysis process model measures. If we know from the user interface assessment that the Class List report contains fourteen data elements then all of the non-composite process model measures can

be determined immediately, as shown in Table 4.11.

| Measure | Value |
|---|---|
| FI | 4 |
| FO | 1 |
| DSP | 4 |
| PDS | 4 |
| DSC | 0 |
| CDS | 0 |
| DEP | 14 |
| DEC | 0 |
| PPI | 0 |
| PD | 3 |
| EEP | 0 |
| PEE | 0 |
| EEC | 1 |
| CEE | 1 |

Table 4.11: Example process model primitive measures

### 4.3.6 Data Model Measures

Measures concerned with the size and interconnection of the data model representation are also included in the analysis scheme. The size of a data model will provide a first-cut, basic indication of the amount of processing that is to be performed on it; that is, a larger data model implies a greater degree of processing to reference, manipulate and/or write to the entities and individual attributes involved. Data model size may also be influential in the estimation of maintenance tasks, particularly for data retrieval systems, where the structure of the existing data will have an impact on how new data should be incorporated into the system. This approach is similar to the work of Symons [236] and Gray *et al.* [102] that was examined in the previous chapter. Measures from this class in the current analysis scheme, as defined in Tables 4.12 and 4.13, include EDM, TESDM, EPD, TDEPD, ECD, TDECD, (T)EP, (T)EC, (T)AU, (T)AC and (T)EL.

The number of entities involved in system operations are considered by the EDM, TESDM, EPD, TDEPD, ECD, TDECD, (T)EP and (T)EC measures. The first two measures are quite straightforward—they are simply counts of the number of entities that are referenced or traversed by a primitive function or by a system. The other measures then consider the types of references that the entities undergo, whether providing data, consuming data or both, as a result of system operations. Use of the measures is based on the same approach as that used for the process model assessment. That is, EPD through to TDECD consider the number of distinct entities referenced, whereas (T)EP and (T)EC consider the actual number of references.

| Microanalysis data model measures | |
|---|---|
| EDM | Number of entities in the data model primitive |
| EPD | Number of distinct entities providing primitive function data |
| EP | Number of entity provisions |
| ECD | Number of distinct entities consuming primitive function data |
| EC | Number of entity consumptions |
| AU | Number of attributes updated by the primitive function |
| AC | Number of attributes consumed by the primitive function |
| EL | Number of entity look-ups performed by the primitive function |
| OOL | Number of 1:1 links between entities in the data model primitive |
| OML | Number of 1:$n$ links between entities in the data model primitive |
| MML | Number of $n$:$m$ links between entities in the data model primitive |
| OL | Number of optional links in the data model primitive |
| ML | Number of mandatory links in the data model primitive |
| MEL | Number of exclusive links between entities in the data model primitive |
| *EA | Entity Access (EP+EC) |
| *AM | Attribute Manipulation (AU+AC) |
| *DMA | Data Model Access (EA+EL) |
| *IDM | Interconnection (Data Model) (OOL+OML+MML) |
| *SDM | Size (Data Model) (EDM+IDM) |

Table 4.12: Primitive function level data model measures

Gray *et al.* [102] suggest that data model measures should also include some consideration of the amount of data actually passed to and from the database. They therefore propose that this could be derived from the number of attributes flowing in the system—for create and delete operations this would be the number of attributes in the entity referenced; for the update and read operations it would be the number of actual attributes referenced. Although this suggestion was developed independently of the current study, these measures equate precisely to the attributes-updated and attributes-consumed measures ((T)AU and (T)AC) of the current proposal. The entity look-up count, (T)EL, should be incremented only when an entity is referenced purely for validation purposes, that is, when an entity is read only to ensure that a particular field value is allowed—the entity's data is not actually used in the process. An entity may, however, be counted more than once for a given system/primitive function if it is accessed both to supply data for processing and for look-up validation.

The interconnection among entities using various relationship types also gives an indication of processing requirements. For example, a one-to-many relationship suggests a hierarchical link, such as that for orders and order lines, providing an insight as to how the relevant data will be entered and processed. Eglington [73] suggests, in fact, that the complexity of many data processing systems is mainly contained in the relationships between records. It would therefore seem worthwhile to consider entity relationship link types at the logical level in an assessment of complexity. Participation requirements may also be important. A mandatory connection, for

example, may indicate a need for validation during processing to ensure that no null entries are supplied. Bushell [38] and Keuffel [140] both state that many-to-many ($n$:$m$) relationships are difficult to implement. Bushell [38] also suggests that connections between entities should be minimised because, if there are several ways of traversing system data for essentially the same purpose, different paths will be used in different cases. Moreover, subsequent changes will be made more difficult. The existence of only one path therefore ensures a standard approach. To this end, the interconnection measures OOL through to TMELS are also included in the analysis scheme.

| Macroanalysis data model measures | |
|---|---|
| TESDM | Total number of entities in the system data model |
| TDEPD | Total number of distinct entities providing data |
| TEP | Total number of entity provisions |
| TDECD | Total number of distinct entities consuming data |
| TEC | Total number of entity consumptions |
| TAU | Total number of attributes updated by the system |
| TAC | Total number of attributes consumed by the system |
| TEL | Total number of entity look-ups performed by the system |
| TOOLS | Total number of 1:1 links between entities in the system data model |
| TOMLS | Total number of 1:$n$ links between entities in the system data model |
| TMMLS | Total number of $n$:$m$ links between entities in the system data model |
| TOLS | Total number of optional links between entities in the data model |
| TMLS | Total number of mandatory links between entities in the data model |
| TMELS | Total number of exclusive links between entities in the data model |
| *TEA | Total Entity Access (TEP+TEC) |
| *TAM | Total Attribute Manipulation (TAU+TAC) |
| *TDMA | Total Data Model Access (TEA+TEL) |
| *TIDM | Total Interconnection (Data Model) (TOOLS+TOMLS+TMMLS) |
| *TSDM | Total Size (Data Model) (TESDM+TIDM) |

Table 4.13: System level data model measures

Derivation of the OOL, OML, MML, TOOLS, TOMLS and TMMLS counts may not be obvious when it comes to the assessment of certain relationship types. Recommendations are therefore made for the following situations:

- a. recursive relationships – as these have no direct impact on the difficulty of development, they should be assessed in the same way as any other relationship
- b. multiple relationships –
    1. where more than one distinct relationship exists between two entities on a given system or primitive data model, and where the system/primitive function being assessed may traverse any of these relationships at one time, each relationship should be counted separately as part of the assessment;

2. where more than one distinct relationship exists between two entities on a given system or primitive data model, and where the system/primitive function being assessed may traverse only a subset of those relationships at one time, each relationship in the subset should be counted as part of the assessment.

Returning to the primitive function depicted in Figures 4.4 and 4.5, the micro-analysis data model measures can now also be determined. For simplicity's sake let us state that each of the entities in the data model primitive contains five attributes and that none of the entities are referenced for look-up purposes. The non-composite data model primitive measures are therefore assigned the values shown in Table 4.14.

| Measure | Value |
|---|---|
| EDM | 4 |
| EPD | 4 |
| EP | 4 |
| ECD | 0 |
| EC | 0 |
| AU | 0 |
| AC | 20 |
| EL | 0 |
| OOL | 0 |
| OML | 3 |
| MML | 0 |
| OL | 2 |
| ML | 4 |
| MEL | 0 |

Table 4.14: Example data model primitive measures

## 4.4 Proposal Summary

Use of the GQM and Classification paradigms at the beginning of this chapter provided a foundation for the determination of the data items required to achieve the goals and objectives of this study. This foundation in turn led to the selection of specific software product and development process measures, based on intuitive expectations and on literary support. Empirical evidence of important measures would clearly have been more reliable; however, the absence of previous studies of this kind means that intuitive expectations have to suffice. The work of Tate and Verner [243], however, has provided indirect but extensive support for the approach adopted in the proposal. They suggest that, in a CASE environment, size measures taken from DFDs, ERDs and user interface representations will be important in estimating other attributes at later development phases. Several possible measures were put forward by Tate and Verner [243], many of which also appear in the current

proposal—for example, numbers of processes and flows, numbers of entities and attributes and numbers of screens and reports. Tate and Verner [243] then raise the question—are all these measures really needed? Their answer, that it is not easy to choose from the measures until some observational work has been performed, provides indirect motivation for the current study. It is also a reflection of the somewhat exploratory nature of this study—it is therefore hoped that the validation of the current scheme will provide recommendations for further research as well as for practical project management.

Now that the analysis scheme has been formally proposed, the next two chapters are concerned with its evaluation. Chapter 5 considers the theoretical validity of the study, in the light of recent discussions on the validity of software measurement as an analysis approach. This is followed by a discussion of the empirical procedures to be used in the evaluation of the proposal, including details of the study's objectives in operational terms, as well as a more in-depth description of the required project management data. Chapter 6 then contains the results of the empirical evaluation and provides a discussion of the findings.

# Chapter 5

# Theoretical Validity and Empirical Procedures

## 5.1 Introduction

If the results obtained from the statistical examination of the analysis scheme are to be used with confidence within the software development industry then both the scheme and the statistical procedures must be shown to be valid and appropriate. This chapter is therefore concerned with the validity and evaluation of the proposed scheme. A discussion on the theoretical validity of the approach is followed by an examination of the criteria used in the empirical evaluation. An outline of the systems analysed in the study is then provided, with the remainder of the chapter being taken up by a discussion of the statistical techniques employed.

## 5.2 Theoretical Validation

Quite distinct from empirical validation, theoretical validation has become increasingly important in its own right over the last five years. Several analysis models and approaches have been criticised for having very weak theoretical foundations. It has been suggested that this failing overrides the validity of any empirical results, as the results are derived from models that are based on flawed assumptions. Issues concerning the validity of software complexity assessment in general are therefore examined in the next section. This is followed by a discussion on the theoretical validity of the current study.

### 5.2.1 Theoretical Validity of Software Measurement

The metric/measurement approach to software complexity analysis has recently received extensive criticism, relating particularly to the assumed equivalence of psychological and structural complexity and to the inadequate internal validation performed in measurement studies. Fenton [79] provides a detailed discussion on the underlying theory and effective use of software metrics. A distinction is made be-

tween *internal* and *external* software attributes, the former being directly derivable from a software product—for example, length or structuredness—and the latter being at least partially dependent on the environment—for example, software reliability or understandability. It is suggested by Fenton [79] that a major failing of many previous studies is the inherent implication, or explicit suggestion, that internal attributes may be used to effectively measure external ones, for example, that code length or design module structure may be used to measure software understandability. (Further discussion of this problem can be found in Fenton and Melton [81] and in Baker *et al.* [8].) Melton *et al.* [177] suggest that this is a result of the failure of most researchers to distinguish clearly between psychological measures and software product measures. Many product measures are said to quantify understandability or maintainability, despite the fact that there are undoubtedly factors other than document structure that affect these attributes. Moreover, any consideration of psychological complexity should incorporate not only the software product but also the person who is attempting to comprehend it. This second component, however, is almost always disregarded because of the difficulty of measuring human understanding.

A second related problem hindering the effective measurement of software complexity is a lack of operational definition. Shepperd and Ince [224] state that most metrics are overgeneralised, in that they are simply expounded as measuring complexity or quality, despite the fact that these attributes are seldom defined beforehand in terms of software development. This has resulted in a situation where 'complexity' has been illustrated in a variety of guises, for example, the number of development errors, the frequency of changes or the effort required to perform an enhancement (Kitchenham *et al.* [153]). Melton *et al.* [177] also remark that units are only rarely provided for the measures extracted, increasing the scope for misinterpretation of results. Moreover, Fenton [79] suggests that it seems curious that so much effort has been invested in validating metrics through the prediction of errors, changes and so on, when definitions of these attributes vary significantly across studies and are seldom provided in the experimental reports.

Inadequate validation has also been cited as a significant drawback of previous metrics research. Fenton [79] separates measurement validation into the two classes of internal and external, with the former being performed far less frequently than the latter. Internal validation should be performed to ensure that a measure is in fact a numerical representation of the property that it claims to quantify. Each measure must be derivable from a clearly defined aspect of the software development process or product and should itself have a formal definition, to ensure that no ambiguity exists in quantification. Fenton [79] cites lines of code (LOC) as an example of an internally valid measure. It is based on program code, a specific software product, and it can be formally defined so that there is no ambiguity in its assessment. The measure is also always able to show if one code example is longer than another, for all code examples. Lines of code is therefore an internally valid measure of code length. It is not, however, an internally or externally valid measure of complexity.

External validation is the process by which an internal attribute, that is, one that is completely derivable from a software product, may be shown to be an important

indicator of some external attribute. This is the type of validation usually performed in metric research. If an internal measure, such as the number of predicate constructs in the code, is validated by relating it to, say, the number of reported post-delivery errors, then it would be fair to say that the measure is a validated indicator of post-delivery error-proneness, but again, not of complexity. This type of validation has always taken preference over that which has investigated characteristics such as quality or reliability because of the difficulty of objectively measuring external attributes such as these. Internal validation, however, is at least as important, and should be performed as a standard component of measurement research (Fenton [79]).

There has also been some discussion of the use of formal axioms as a basis for the development of appropriate measures (Prather [198]; Bollmann and Zuse [28]). These studies suggest that the use of such axioms will ensure that no easily confounded metrics will be proposed, as they will fail to satisfy the axioms. However, Cherniavsky and Smith [47] showed that the axiomatic-type approach developed by Weyuker [260] could easily be circumvented by a nonsensical measure. Shepperd and Ince [225] have also criticised the Prather approach [198] for its operational weakness. Moreover, all of the measurement axioms suggested so far have been applicable only to software code. With changes in technology this would appear to have little current applicability, at least within the commercial software development domain.

### 5.2.2 Validity of the Current Study

The approach adopted in this study has attempted to conform as closely as possible to the recommendations and remarks made in the previous section. Whereas many previous measurement proposals have asserted to quantify understandability or psychological complexity, this is certainly not a claim of this project. In fact, it is seen as an advantage of this proposal that understandability is of much less influence, due to automatic system generation facilities and English-like language use. Where acknowledgement is made of the need for specification understanding, for example, in software maintenance, it should be clear that only functional complexity, and not understandability, is considered to be assessable. Thus this study holds no claim to being able to measure software understandability or psychological complexity. Some reiteration may therefore be required as to the overall objective of the study, that is, the development and validation of a specification-based *functional complexity analysis or assessment* scheme applicable to interactive commercial systems.

**Analysis or assessment** – throughout the discussion of the proposed scheme in the previous chapter it was referred to as an analysis or assessment scheme and not as a measurement scheme. Consider the following dictionary definitions (Allen [5]):

**analyse** – examine in detail; ascertain elements or structure of [complexity];

**analysis** – detailed examination of elements or structure of [complexity];

**assess** – estimate magnitude or quality of [complexity];

**measure** – (*v.*) find extent or quantity of [complexity] by comparison with fixed unit or with object of known size.

Measurement therefore requires the use of fixed units or well-defined measurable baseline properties. Given that complexity is abstract, multi-dimensional and poorly defined, scientific measurement would appear to be an unlikely prospect. Analysis or assessment, on the other hand, can be performed without the need for a more solid definition of the item being considered, which is in this case functional complexity.

**Functional complexity** – despite widespread acknowledgement of the absence of an operational definition, 'complexity' *is* still used here. Although this does not adhere totally to the comments of the preceding section, continued use of this expression is considered to be acceptable, for three reasons. Firstly, it is almost certain that we will never have a universally applicable measure for software complexity, if only because of constantly changing technologies. This should not, however, preclude us from using the term as a *descriptive indication* of the effect of a combination of (what are perceived to be) important factors that make a given task in any domain, not just software development, more difficult to perform.

Secondly, a lack of complete definition should also not stop us from making *value judgements* regarding relative levels of certain attributes, in spite of the fact that the attributes themselves have not been measured. For example, a student may remark: 'Today's test was much more difficult than the one we had last week'. Clearly this is based on individual perception and on other measurable factors, such as the time needed to complete the test, the number of questions in the test, the number of questions completed and so on. Although difficulty itself has not been measured, various levels of the attribute may be compared in general terms based on other related measures, such as the number of questions completed. It is considered that, at least in this particular study, a similar approach may be applicable for complexity analysis.

The third reason for the continued focus on complexity is its multi-dimensional nature. Measures of an attribute that may be much more clearly defined, such as measures of size, account for only one of the many aspects of software that may contribute to the likelihood of errors and to development effort requirements. Other features, including interconnection, are ignored by size indicators. This can result in a situation in which size measures are only poor discriminators of actual complexity, one of the failings of some of Halstead's measures and the lines-of-code measures. Thus complexity is not necessarily a component of size. However, complexity indicators can include measures of size, as well as other measures, so that a comprehensive analysis of all the contributing factors may be performed.

Thus, although not measures of complexity itself, a basic assumption of this study is that the various specification measures are certainly *related* to complexity. Oman and Cook [194] state that, in general, complexity measures do not measure complexity itself, but the extent to which those features thought to contribute to complexity exist in a software product. The specification measures investigated in this study are therefore considered to be *indicators* of functional complexity and the project management attributes are viewed as the partial *consequences* of that functional complexity. It should be noted, however, that given the abstract nature of complexity, the link between these measures and complexity is one based on intuitive expectations and not on any validated direct mapping of the measures to complexity. Thus although this study may provide empirical evidence of relationships between the specification measures and the project management attributes, the

link between these relationships and 'complexity' is founded only on the assumption that systems/functions returning higher value specification and project management measurement values are more complex than those returning lower values.

The specification measures considered here are all internally valid according to Fenton's criteria [79]. All are derivable from well-defined abstractions of software products, that is, ERDs, DFDs, FDHs, screen and report formats and data dictionaries. All may be rigorously defined to ensure that counting of any item is unambiguous, and all quantify relative levels of product attributes, for example, the number of entities in a data model primitive or the maximum depth of a function. Given unambiguous definitions and assessment procedures, the project management indicators, as discussed in the following section, should also be considered to be internally valid.

## 5.3 Empirical Validation

It was stated in Chapter 1 that more than ninety complexity metrics are currently in existence. One of the reasons that many of these methods have not been used in the development industry is a lack of 'proof' that they actually provide some form of consistent benefit to prospective users. In cases where it is provided, this proof is most often an analysis of the results obtained by applying the measurement scheme to a sample of representative systems. Although this type of verification is clearly to be encouraged, problems have occurred with this activity, particularly in relation to the use of inappropriate statistical techniques when undertaking data analysis. The following sections therefore describe the empirical validation and analysis procedures used in the current study. The project management criteria used in the validation are explained in the next section, followed by a short description of the systems analysed and a discussion of the statistical techniques employed.

### 5.3.1 Evaluation Criteria

As stated previously, complexity is thought to be influential in determining two types of development attributes:

1. attributes such as software quality and reliability;

2. attributes such as development effort and error occurrence.

Given the difficulties in obtaining objective, quantitative indications of the items in the first category, evaluation in this study is restricted to the discrimination and estimation of attributes from the second class.

In the previous chapter it was suggested that high values of the various specification measures would indicate primitive functions/systems that were time consuming and error-prone to develop. To empirically evaluate the proposed analysis scheme, this assertion must be tested—quantitative indicators of development effort and error occurrence are therefore required at both levels of analysis. For the purposes of this study, these indicators are defined as follows:

- analysis effort – time, in person-days, spent on analysing and specifying the requirements of a primitive function/system in an automated environment

- design effort – time, in person-days, spent on designing a primitive function/system in an automated environment
- construction effort – time, in person-days, spent on constructing a primitive function/system in an automated environment
- unit test effort – time, in person-days, spent on the unit testing of a primitive function/system in an automated environment
- system test effort – time, in person-days, spent on the integration testing of a system in an automated environment
- total development effort – time, in person-days, spent on analysing, designing, constructing and testing a primitive function/system in an automated environment
- number of errors – number of functional errors applicable to a primitive function/system reported during the acceptance testing phase
- number of amendments – number of functional amendments applicable to a primitive function/system reported during the acceptance testing phase.

These indicators are related to various well-supported assumptions associated with relative complexity levels (Gremillion [103]; Brooks [32]). It is generally expected that a more complex primitive function/system will take longer to develop and test than a less complex counterpart. The effort measures therefore reflect the amount of work carried out by personnel using CASE tools and/or 4GLs over the various phases of development. It is also likely that a complex primitive function/system will be more prone to errors during its development and will require a greater number of amendments after initial delivery, due to increased misunderstanding between users and developers. Thus the error and amendment counts are included. Errors are defined to be instances where the required functionality, as represented in the specification, is missing or has been incorrectly implemented. Amendments, on the other hand, represent situations in which the functionality is present and is performed correctly by the system, but in a different way than that required by the user.

All of these project management indicators can still be influenced in unexpected ways by other factors, including organisational changes. However the impact of several external influences, including those relating to personnel, will be reduced within an automated development environment, so the overall influence of outside factors should be lessened. It is certainly an assumption of this study that extensive quantitative analysis is currently the best method available for obtaining early indications of effort requirements and error occurrence, despite the possible influence of factors that cannot be anticipated. Thus the general expectation of the evaluation is that primitive functions or systems that return higher specification complexity indicator values will be more time-consuming and error-prone to develop than primitive functions or systems that return lower indicator values.

### 5.3.2 Systems Analysed

After an extensive mailing campaign, ten business and government organisations subsequently agreed to provide systems for this project. Most agreed to allow one system only to be analysed, giving an overall sample size of sixteen systems. A much larger sample of both sites and systems had been anticipated at the beginning of this research but the response from development sites was extremely disappointing. Although the sample is certainly small, it is hoped that the results will be applicable to systems developed with a wide range of CASE and 4GL tools, given the variation in products examined. Appendix A.1 contains further discussion on the mailing campaign and on the resulting response from development organisations.

The ten organisations that agreed to participate in the study are as follows:

- BP Chemicals Ltd – a subsidiary of one of the United Kingdom's largest companies, manufacturing chemical and plastic products for a worldwide market
- British Gas plc – concerned with the exploration, purchase, distribution and sale of gas in the U.K. and overseas, supplying over 17.5 million domestic sites
- Home Office – a government department concerned with the administration of justice, immigration and public safety in the United Kingdom
- ICI Chemicals and Polymers Ltd – a subsidiary of Imperial Chemical Industries plc, involved in the manufacture of chemical products for the European market
- Merrett Management Services Ltd – a subsidiary of Merrett Holdings plc, providing accounting and personnel services for Lloyd's underwriting and insurance agencies
- Office of Population Censuses and Surveys – the government department responsible for the registration and reporting of demographic statistics in the United Kingdom
- Pro IV Holdings Ltd – a private company providing information systems to a national client base that includes airlines, breweries and local government bodies
- Rover Advanced Technology Centre – an industry-sponsored research group undertaking projects of interest to the motor vehicle industry
- Royal Insurance (UK) Ltd – one of the U.K.'s largest insurers, providing almost all types of insurance services to a national market
- Unipart Information Technology – designers, manufacturers and distributors of automotive parts, components and accessories for the U.K. market.

The sixteen systems in the sample performed a number of overall functions, including customer and supplier recording, costing and charging, accounting, site and personnel administration, scheduling and rostering, and were implemented on

a range of mainframe and microcomputer platforms. CASE tools and/or 4GLs were used extensively in the development of all sixteen systems. The tools used included Oracle CASE, AutoMate Plus, IEW/ADW, Model 204 and MADM, the IEF, Quickbuild, Excelerator, Application Master, ProKit Workbench and Pro IV.

Collection of the analysis scheme data items was performed manually from various specification documents, that is, ERDs, DFDs, FDHs, screen and report formats and data dictionaries. Although automatic extraction would have been more efficient and less error-prone, the tools used in the development of the systems investigated here did not have the facilities to perform this function. Furthermore, manual collection also resulted in a minimum of interruption to the normal operations of the organisations, whereas in-house analysis without automatic extraction tools would have been more disruptive. The project data relating to development effort and reported errors was gathered from a combination of on-line and paper-based records that had been kept as part of the organisations' routine project management procedures.

### 5.3.3 Statistical Analysis Techniques

Analysis procedures were chosen so that the empirical objectives of the study, as stated in Chapter 1 and repeated here, could be achieved. The original objectives were as follows:

- the determination of relationships between functional complexity indicators and project management data (relating to development effort and error occurrence)
- the early determination of relative functional complexity indicators (in terms of development effort and error occurrence) at both the system and individual function level
- the classification of systems and individual functions according to their likely project management consequences (in terms of development effort and error occurrence) based on functional complexity indicators
- the development of equations for the estimation of project management data (relating to development effort and error occurrence) based on functional complexity indicators.

In order to satisfy the above objectives a number of statistical techniques were used. Recent work in the software measurement domain has highlighted a need for the use of more appropriate statistical procedures in the analysis of collected data (Ince and Shepperd [125]; Coupal and Robillard [57]). The distributions of complexity analysis data, both product measures and project management measures, are often skewed to the right and contain a number of outliers. This is particularly the case for primitive function error data, which can never take a value of less than zero and yet are often concentrated near the zero data point. Similarly for product measurement, data points tend to be clustered at the lower end of the distribution

(Kitchenham [150]; Card *et al.* [39]). A number of previous studies have failed to take this into consideration when using statistical analysis techniques. Therefore in this study a variety of procedures were used in an attempt to ensure that the results obtained were valid and that they provided a sound basis for the development of appropriate and applicable conclusions. A short description of the procedures now follows.

### Correlation

The Pearson product-moment correlation coefficient indicates the extent of a linear association between two variables (Woodfield *et al.* [263]). The value of the coefficient can take a value between –1 and +1, where a value approaching –1 indicates an increasingly negative, or inverse, linear relationship, and a value approaching +1 indicates an increasingly positive or direct linear relationship. A coefficient value of 0 indicates that the two variables are unrelated. Use of this coefficient requires that the underlying data should have been drawn from normally distributed samples (Knafl and Sacks [155]).

A correlation technique that does not make this assumption of normality is the Spearman rank correlation coefficient. This is a measure of the correspondence between two sets of observations when they are ranked in the same order; it is also considered to be an indication of the strength of the relationship between the two variables of interest (Daniel [62]). Thus the measure can illustrate the relative correspondence between two variables, as required by the second objective. The Spearman statistic is subject to the same limits as the Pearson coefficient. The only drawbacks to the use of this statistic are that it can be confounded by large numbers of tied data elements and that a correspondence of ranks does not *necessarily* reflect a close linear relationship (which the Pearson statistic does indicate). Thus an examination of the data should be carried out in conjunction with the computation of the correlations.

### Classification

Classification and outlier identification techniques have become increasingly popular within software assessment procedures over the last decade. Kafura and Canning [134], Card *et al.* [39] and Kitchenham *et al.* [153], for example, describe a number of methods for component classification and for the detection of abnormal modules. In cases where data follows the normal pattern, aspects of the distribution can be used to examine the effectiveness of, for example, a specification measure in classifying systems or functions according to their subsequent requirements for development effort and error-proneness, as performed in this study. They can also be used in the identification of relatively abnormal 'outlier' values. Those systems or functions with abnormally high values of influential specification measures may demand further development or enhancement to ensure that problems do not arise in subsequent development phases (Shepperd and Ince [224]; Kitchenham and Pickard [152]). The method employed in this study used a direct comparison of the distributions of values

for the two related variables, as initially identified by the correlation techniques described above.

For example, fifteen systems may return the following set of values for measure $A$: 7, 9, 13, 11, 14, 14, 19, 21, 15, 20, 23, 26, 26, 30 and 28. The arithmetic mean of this distribution is 18.4 and the standard deviation (s.d.) is 6.955. For those same fifteen systems, measure $B$, to which measure $A$ is related, takes the values: 94, 106, 108, 119, 111, 102, 122, 125, 119, 118, 141, 136, 140, 144 and 151. The mean in this case is 122.4 and the standard deviation is 16.337. If we consider an outlier to be a data element that has a value greater than one standard deviation above the mean, given the small size of the data sets, an analysis such as the following may be performed:

```
Measure A:
mean = 18.400, mean - 1 s.d. = 11.445, mean + 1 s.d. = 25.355

                        11.445          18.400          25.355
Systems:  1 2 4           | 3 5 6 7 9     | 8 10 11       | 12 13 14 15
                          |               |               |
                          |               |               |

Systems:  1 2 6           | 3 4 5 7 9 10  | 8 12          | 11 13 14 15
                          |               |               |
                          |               |               |
                       106.063         122.400         138.737

Measure B:
mean = 122.400, mean - 1 s.d. = 106.063, mean + 1 s.d. = 138.737

Direct mapping for systems: 1 2 3 5 7 9 8 13 14 15
Number of systems: 15      Correct classification: 10/15 = 66.7%

Direct mapping for outliers: 13 14 15
Number of response outliers: 4     Correct identification: 3/4

Number of incorrectly identified outliers: 1
```

Thus in the example above the classification of a system according to the categories of measure $A$ would also be the correct classification for the system in terms of measure $B$, for 66.7% of the systems classified. So if, for example, we were to state that measure $A$ was the number of entities in the system data model and that measure $B$ was the total number of development effort hours, such a technique would enable a project manager to estimate with 66.7% confidence, within two bounding figures, the approximate number of development hours that a project would require based only on the number of entities in the system data model. Furthermore he or she could predict with 75% accuracy the systems that would be outliers in terms

of development effort. A significant advantage of this approach is that it can be performed entirely at the specification stage.

If the data in question is not normally distributed the classification procedure could instead be carried out based on boxplot distributions. Boxplots (Hoaglin *et al.* [120]), such as the one shown in Figure 5.1 (Norusis [193]), depict the spread of values about the median of a distribution. The median is considered to be a more robust indicator of central location when the underlying data is skewed (Rousseeuw and Leroy [211]; Daniel [62]; Norusis [193]).

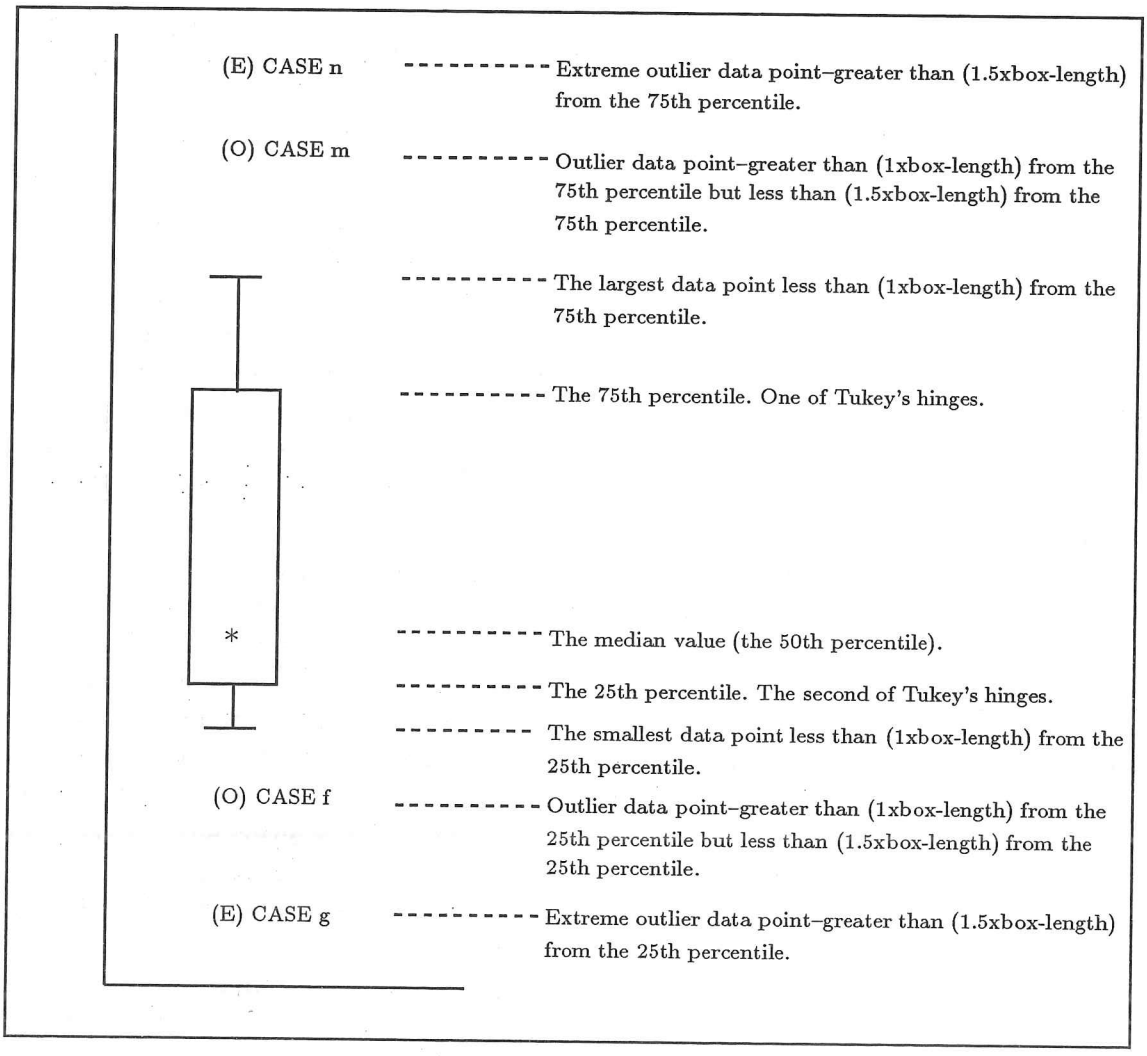



Figure 5.1: Boxplot diagram

Classification using boxplot distributions is also performed according to four bounded categories. The lower class contains those data points with values less than the median value; the second class includes items with values greater than or equal to the median but less than that of the 75th percentile; the third group of values must be greater than or equal to that of the 75th percentile but less than the upper whisker value; and the fourth class contains all those data points with values greater than or equal to the upper whisker value. This fourth class therefore

contains all the outliers of a given distribution.

### Estimation

Accurate prediction of requirements and outcomes for projects still to be developed has been one of the most sought after goals of software assessment research. This is particularly the case for the estimation of effort—models have been proposed by, among others, Albrecht [2], Symons [236] and Boehm [25]. Two factors that are common to most, if not all, effort estimation methods are the need for extensive calibration and the assessment of a number of external, that is, non-system related considerations. Both procedures are considered to be essential if the results obtained are to be applicable to the local development environment. One of the purposes of this study, however, was to develop an assessment method that was less dependent on both development personnel and on aspects particular to individual development organisations or groups. It is therefore envisaged that calibration for 'unique' circumstances will not be as necessary. If a relatively reliable equation can be obtained from the samples used here, with the variations in sites and products employed, then approximate predictions using a single equation may become more feasible.

Another important reason for the inclusion of estimation procedures in this study is less justifiable in theoretical terms, but is important in a more pragmatic manner. Although effective predictions are difficult to obtain and are not *necessary* for effective software measurement (Baker *et al.* [8]) they are almost certainly what project managers require. Therefore if this study can provide equations that are useful for the majority (say, seventy-five percent) of MIS and transaction-based projects using automated tools in committed development organisations then this will still be a worthwhile and potentially useful achievement.

The popular least-(mean-)squares (LS) regression method was therefore used in conjunction with the less common least-median-squares (LMS) technique in an attempt to ensure that robust estimates, that is, estimates that are not overly influenced by outliers, are developed for the predictions in this study. The LS method has become the cornerstone of classical statistics, due to both ease of computation and to tradition (Rousseeuw and Leroy [211]). In cases where outliers seldom occur, the LS method is often more than adequate. However, outliers are a common feature of software engineering data sets (Kitchenham and Pickard [152]). Moreover, the LS technique yields the arithmetic mean of observations, in spite of the fact that, for skewed data, the mean is not a robust indicator of central location. Thus even in cases where there are small departures from the normal model the LS method loses much of its efficiency (Hampel *et al.* [109]; Myrvold [184]). The LMS method, as discussed by Rousseeuw and Leroy [211], was therefore also used.

The traditional statistic for assessing the strength of a linear relationship under the LS method is the coefficient of determination ($R^2$). This statistic reflects the amount of variation in the response variable that can be explained by the predictor variable(s). It takes a value of between zero and one, with the latter value indicating a total ability of the predictor variable to explain the variation of the response variable. An analogous statistic, also called the coefficient of determination, is available

for the LMS regression, although the formulation is slightly different. This coefficient was used in this study to assess the effectiveness of the regression equations, in conjunction with examinations of the residuals and, in appropriate cases, an $F$-test of the significance of the regression slope.

### Computer-Based Analysis

The set of techniques described above includes methods applicable to normally and non-normally distributed data sets. Particular statistical methods were deemed to be applicable to the various data sets according to two tests of normality—the Lilliefors version of the Kolmogorov-Smirnov test and the powerful Shapiro-Wilks test (Daniel [62]; Norusis [193]).

The methods were implemented using two different computer-based statistical analysis packages. The correlation procedures and the distribution/normality tests were performed using the SPSS/PC+ system. Both the Spearman and Pearson correlation coefficients were provided to two levels of significance, enabling the determination of appropriate relationships and the selection of variables for subsequent examination. The distribution and normality tests also automatically provided significance figures so the selection of classification methods was straightforward.

The regression tests were carried out using the PROGRESS (Program for RObust reGRESSion) package developed by Rousseeuw and Leroy [211]. The PROGRESS system includes both the least-squares and least-median-squares techniques, and automatically computes reweighted least-squares (RLS) equations based on the LMS analysis results. The RLS procedure removes the outliers identified in the LMS regression and computes a new LS equation based on the remaining data points. The PROGRESS system also enables the examination of residual plots, so as to ensure that any equation developed satisfied the requirements for residuals, that is, that they are independent of one another, that they are evenly dispersed about the mean (at zero on the vertical axis) and that they reflect a constant variance. Linear prediction models that produce residuals that do not conform to these requirements are generally inadequate, in that they may be improved only through the inclusion of weighted or transformed terms.

The methods described in this section were chosen in order that the objectives of the study could be achieved in a valid manner. Both parametric and non-parametric correlation techniques were used to satisfy the first two objectives, boxplots and normal distributions were used to carry out the classifications required by objective three and both least-mean- and the more robust least-median-squares regression methods were applied so that the fourth objective could be achieved. The results obtained from the application of all these procedures are summarised in the next chapter. Examples of the actual listings that were produced as output from the two packages appear in Appendix A.2.

for the LMS regression, although the formulation is slightly different. This coefficient was used in this study to assess the effectiveness of the regression equations, in conjunction with examinations of the residuals and, in appropriate cases, an $F$-test of the significance of the regression slope.

### Computer-Based Analysis

The set of techniques described above includes methods applicable to normally and non-normally distributed data sets. Particular statistical methods were deemed to be applicable to the various data sets according to two tests of normality—the Lilliefors version of the Kolmogorov-Smirnov test and the powerful Shapiro-Wilks test (Daniel [62]; Norusis [193]).

The methods were implemented using two different computer-based statistical analysis packages. The correlation procedures and the distribution/normality tests were performed using the SPSS/PC+ system. Both the Spearman and Pearson correlation coefficients were provided to two levels of significance, enabling the determination of appropriate relationships and the selection of variables for subsequent examination. The distribution and normality tests also automatically provided significance figures so the selection of classification methods was straightforward.

The regression tests were carried out using the PROGRESS (Program for RObust reGRESSion) package developed by Rousseeuw and Leroy [211]. The PROGRESS system includes both the least-squares and least-median-squares techniques, and automatically computes reweighted least-squares (RLS) equations based on the LMS analysis results. The RLS procedure removes the outliers identified in the LMS regression and computes a new LS equation based on the remaining data points. The PROGRESS system also enables the examination of residual plots, so as to ensure that any equation developed satisfied the requirements for residuals, that is, that they are independent of one another, that they are evenly dispersed about the mean (at zero on the vertical axis) and that they reflect a constant variance. Linear prediction models that produce residuals that do not conform to these requirements are generally inadequate, in that they may be improved only through the inclusion of weighted or transformed terms.

The methods described in this section were chosen in order that the objectives of the study could be achieved in a valid manner. Both parametric and non-parametric correlation techniques were used to satisfy the first two objectives, boxplots and normal distributions were used to carry out the classifications required by objective three and both least-mean- and the more robust least-median-squares regression methods were applied so that the fourth objective could be achieved. The results obtained from the application of all these procedures are summarised in the next chapter. Examples of the actual listings that were produced as output from the two packages appear in Appendix A.2.

# Chapter 6

# Empirical Analysis Report

## 6.1 Introduction

The two previous chapters describe in detail the proposed analysis scheme and the methods that were used in its evaluation. These factors are now combined in this, the report on the statistical analysis of the scheme. The chapter begins with a brief description of the final samples and a short discussion on the availability of the necessary data. This is followed by a summary of the results obtained from the analysis of each of the four samples. Sample results, including listings of the raw data and of the direct output of the statistical analysis procedures, are provided in Appendix A.2. A discussion of the results and appropriate recommendations then conclude the chapter.

## 6.2 Description of Samples

The sixteen systems described in Chapter 5 provided four distinct samples for statistical analysis. Table 6.1 describes the four samples in terms of the number of data sets in each, the level at which the data was collected and the type of project management data that was available for those data sets.

| Sample | Level | Project Data | No. Data Sets |
|---|---|---|---|
| 1 | Macroanalysis | Effort | 13 |
| 2 | Macroanalysis | Errors | 11 |
| 3 | Microanalysis | Effort | 1 |
| 4 | Microanalysis | Errors | 1 |

Table 6.1: Analysis samples

Table 6.1 shows that macroanalysis data was collected from thirteen and eleven projects for the effort and error investigations respectively. Although these are not large samples they are certainly non-trivial, given that they were obtained from a selection of diverse sites. Moreover, very little empirical work has been previously

undertaken concerning complexity assessment at the specification stage—as far as the author is aware there have been no previous studies of this type that have concentrated solely on systems developed with extensive automated assistance. It is therefore felt that the results obtained from the macroanalysis investigations will prove to be useful when applied to future development projects.

Unfortunately this is not likely to be the case for the microanalysis samples. As illustrated in Table 6.1 only one data set was available for the investigation of relationships for both the effort and error perspectives. Although Chapter 4 highlighted the reasons in favour of low-level data collection it would appear that this activity is still not widely practised. With increased automation of collection procedures this situation is likely to change over the next five years—more effective investigations will then be possible. In spite of the clear limitation that this places on the current microanalysis study, statistical analysis should still provide some preliminary insight into the effectiveness of low-level specification assessment. Clearly, however, it would be unwise to draw any generalisable conclusions from such an analysis. Therefore much of the following discussion is based on the analysis of the system level data. Information specific to the single microanalysis data sets has been included with the result summaries for samples three and four provided below.

### 6.2.1 Data Availability

Due to access difficulties, several system level variables of interest proposed in the scheme had to be discarded before the analysis could proceed. From the set of macroanalysis user interface measures only the numbers of distinct screens and reports (TDSCR and TDREP) were consistently available. This was due to the fact that just three of the system specifications contained, in document form, copies of the system screens and reports. The number of report and screen calls and the number of data elements used were therefore only accessible in these three cases. Thus the TREPC, TDER, TSCRC and TDED variables were removed from the analysis.

More significantly, it was decided that the analysis of process model measures would be abandoned at the outset, as only three projects were found to have made full use of data flow diagrams in their system specifications. Although this finding may be taken to suggest that DFDs are now uncommon, it is more likely that it is simply a reflection of the particular tools that were encountered in this study. A recent assessment of several CASE products (Vessey *et al.* [255]) showed that DFD assistants were still among the most widely available features of automated tools. Future studies with systems developed using other tools should therefore provide useful results relating to the influence of process model factors. Only one of the macroanalysis data model measures was removed before the statistical procedures were applied. The number of entity look-ups (TEL) was documented as a separate figure in just two of the projects—in all other cases a look-up was included as a read operation. Thus statistical analysis of the variable was not applicable.

The need to discard the variables mentioned was at least partially due to the fact that data collection in this study was based completely on document versions of specifications. If access to the system-held specifications had been allowed then

many of the discarded variables would have become available. Therefore although they have been removed from this study the variables may still be of interest, and if the counting scheme is automated then variables of this type will be more readily accessible. Analysis of the complete data set may then be performed.

Some of the project management indicators were also removed before the commencement of the statistical analysis. It was found that, not unexpectedly, records of phase effort were not recorded in a consistent manner across both projects and organisations. For example, some sites performed only design before construction, and any analysis-type activities were included as design tasks. Similarly, construction effort often incorporated unit-testing effort, rather than it being recorded as a distinct figure. It was therefore decided to create two new project management indicators:

- analysis-design effort – time, in person-days, spent on analysing, specifying and designing a primitive function/system in an automated environment
- program-unit test effort – time, in person-days, spent on constructing and unit testing a primitive function/system in an automated environment.

As system test effort figures were also only available for four projects it was necessary to omit this variable at the outset. This therefore left five effort indicators—the two defined above, denoted AN_DES and PROG_UT in the statistical discussion, along with the original design effort, program effort and total effort measures, referred to as DESIGN, PROGRAM and TOTAL respectively. In terms of error indicators, both the originally defined measures of errors and amendments were available. These are denoted ERRORS and AMEND in the following discussions.

## 6.3 Analysis Results

The following sections present the results obtained from the statistical examination of the analysis scheme measures and the project management indicators, for each of the four samples described earlier in this chapter. These results form the basis for the subsequent discussions and conclusions of the study.

### 6.3.1 Sample One: Macroanalysis—Effort

A total of thirteen valid project data sets were collected for this sample. Correlation procedures identified a number of highly significant associations involving variables from all of the specification perspectives. Many of these relationships were significant at the $\alpha = 0.001$ level; that is, there was less than 0.1% probability that the relationships had been encountered by chance. Those specification indicators that were found to be significantly correlated with system level effort indicators at this level are shown in Table 6.2. The abbreviation 'C.M.' stands for the correlation method, with 'P' denoting Pearson's statistic and 'S' representing the Spearman statistic.

| Perspective | C.M. | Specification Indicators |
|---|---|---|
| Transaction | P | TRE |
| | S | TRE |
| Functional | P | FLEV L4 L5 TFUNC TD |
| | S | FLEV L5 TD |
| User Int. | P | TDSCR |
| | S | TDSCR |
| Data | P | TESDM TDEPD TEP TDECD TEC TAU TAC TOMLS TOLS TMLS TEA TAM TIDM TSDM |
| | S | TESDM TDEPD TAU TAC TMLS TAM TIDM TSDM |

Table 6.2: Significant correlations—macroanalysis indicators and effort

Since the Spearman coefficient is said to be conservative, except in cases where ties are common (Kitchenham and Pickard [152]), it was decided that further analysis would be carried out only on variables that showed highly significant values for both this statistic and the Pearson statistic. The variables chosen based on this criteria appear in Table 6.3.

| Perspective | Specification Indicators |
|---|---|
| Transaction | TRE |
| Functional | TD |
| User Int. | TDSCR |
| Data | TESDM TDEPD TAU TAC TMLS TAM TIDM TSDM |

Table 6.3: Macroanalysis–effort indicators chosen for further analysis

Thus only two correlated variables were discarded when the choice of variables for subsequent analysis was made. The L5 variable was found to contain a large number of data points with a value of zero, producing a high degree of correlation that was not evident from the data itself. L5 was therefore removed before the next tests were carried out. Similarly, the FLEV variable contained a number of data points of equal value; it was therefore also removed.

Another variable selection method was then employed to ensure that interrelated variables did not go forward for use in the other tests. Kitchenham and Pickard [152] suggest that closely related predictor indicators should be treated with caution when used together, especially for estimation purposes. Furthermore it is often the case that one of a group of interrelated variables is sufficiently powerful to act for the group. Criteria other than the original correlation coefficients should therefore be used to select appropriate independent variables from related groups. In cases where the data is normally distributed some form of factor analysis may be useful. As stated earlier, however, normality in software engineering data distributions is uncommon. Hampel *et al.* [109] suggest that there is practically always no guarantee of normality and that slight departures from the model have a significant effect on

the results obtained. Variables were therefore selected from groups according to their ease of extraction and the time at which they became available—variables that are easily determined and are available as early as possible are clearly to be preferred over more complicated, later-phase variables.

Correlation tables illustrated the significantly high degree of correlation within the chosen group of variables from the data perspective. The data model measures were all very highly correlated, except for the TAU and TAC variables. Since these two variables were easily extracted, were elementary rather than composite, were available very early in the development process and appeared to be relatively independent but still highly correlated with the effort indicators, they were both selected for separate use in the procedures to follow. For the current sample this led to a final set of specification variables, as shown in Table 6.4. A summary of the correlation test results is provided in Table 6.5. This table shows the variables finally chosen and the levels of correlation that were achieved between the specification indicators and the effort variables. All the correlation coefficients were significant at the $\alpha = 0.001$ level.

| Perspective | Specification Indicators |
|---|---|
| Transaction | TRE |
| Functional | TD |
| User Int. | TDSCR |
| Data | TAU or TAC |

Table 6.4: Independent macroanalysis–effort indicators

| Specification Indicator | Effort Indicator | Pearson Correlation | Spearman Correlation |
|---|---|---|---|
| TRE | TOTAL | 0.8058 | 0.6923 |
| TD | TOTAL | 0.8876 | 0.7912 |
| TDSCR | AN_DES | 0.9053 | 0.7180 |
| | TOTAL | 0.7800 | 0.8748 |
| TAC | PROG_UT | 0.8471 | 0.8736 |
| | TOTAL | 0.9160 | 0.9341 |
| TAU | AN_DES | 0.9372 | 0.8077 |

Table 6.5: Macroanalysis–effort variable summary

The descriptions of the classification procedures provided in the previous chapter stated that appropriate methods would be chosen according to the results of normality tests of the distributions of the selected variables. The results of the distribution tests for this sample are summarised in Table 6.6.

| Indicator | Shapiro-Wilks | Lilliefors | Classification Method |
|---|---|---|---|
| TRE | Normal | Normal | Normal |
| TD | Non-normal | Non-normal | Non-normal |
| TDSCR | Non-normal | Normal | Both |
| TAC | Non-normal | Non-normal | Non-normal |
| TAU | Non-normal | Non-normal | Non-normal |
| DESIGN | Normal | Normal | Normal |
| AN_DES | Non-normal | Non-normal | Non-normal |
| PROG_UT | Normal | Normal | Normal |
| TOTAL | Normal | Normal | Normal |

Table 6.6: Macroanalysis–effort normality tests

A normal distribution was considered to be applicable if the level of significance of the Shapiro-Wilks or Lilliefors statistic was greater than $\alpha = 0.01$ for each of the variables. Otherwise a non-normal distribution was considered to be more appropriate. The selected classification method, shown in the final column of Table 6.6, was based on the combination of the two statistical results returned for each variable. Where both statistics returned the same result for a variable then that method, normal or non-normal, was chosen as the appropriate classification procedure. If two different results were returned, however, as in the case of TDSCR in Table 6.6, then both methods were selected.

| Specification Indicator | Effort Indicator | Distribution Results | Classification Method |
|---|---|---|---|
| TRE | TOTAL | Normal | Normal |
| TD | TOTAL | Mixed | Both |
| TDSCR | AN_DES | Mixed | Both |
| | TOTAL | Mixed | Both |
| TAC | PROG_UT | Mixed | Both |
| | TOTAL | Mixed | Both |
| TAU | AN_DES | Non-normal | Boxplot |

Table 6.7: Macroanalysis–effort classification method selections

The information in Table 6.7, which shows the combined results of the distribution tests, formed the basis for the classification tests. In cases where the distributions of both the related specification and effort indicators were the same, the 'Distribution Results' column was filled with the common term, normal or non-normal. Otherwise it was shown as a mixed result. A normal distribution result indicated that classification would be carried out using the parameters of the normal distribution; a result of non-normal suggested a boxplot-based classification. A mixed result indicated that both methods would be used, and the most effective one would then be chosen from them.

| Spec. Indic. | Effort Indic. | Classif. Method | Classif. Correct (%) | Outliers Correct | Excess Outliers |
|---|---|---|---|---|---|
| TRE | TOTAL | Normal | 84.6 | 2/3 | 0 |
| TD | TOTAL | Normal | 76.9 | 2/3 | 0 |
| | | Boxplot | 76.9 | 2/2 | 0 |
| TDSCR | AN_DES | Normal | 84.6 | 1/2 | 0 |
| | | Boxplot | 69.2 | 2/3 | 0 |
| | TOTAL | Normal | 69.2 | 1/3 | 0 |
| | | Boxplot | 92.3 | 2/2 | 0 |
| TAC | PROG_UT | Normal | 84.6 | 2/3 | 0 |
| | | Boxplot | 61.5 | 0/1 | 1 |
| | TOTAL | Normal | 84.6 | 2/3 | 0 |
| | | Boxplot | 69.2 | 0/2 | 1 |
| TAU | AN_DES | Boxplot | 76.9 | 3/3 | 0 |

Table 6.8: Macroanalysis–effort classification results

The results of the classification tests are summarised in Table 6.8. The table shows how effective the specification indicator variables were in correctly classifying the thirteen systems in relation to their final effort requirements. Several results showed a high degree of success in both the classification of systems and in the identification of outliers. Moreover, only two of the classifications incorrectly identified outlier data points. This was clearly a useful outcome of the tests, although the spread of some of the distributions did mean that the classifications provided only approximate estimations of subsequent effort. For example, even though the TDSCR variable returned a 92.3% success rate in total effort classification, the dispersion of the TOTAL data significantly reduced the usefulness of such a result. If a new specification contained, say, fifty-six screens we could only predict that the total effort needed to develop that system would fall somewhere between 226.5 and 315 days, an unacceptably wide time interval in terms of effective schedule management.

Based on the results obtained from both the correlation and the classification tests a set of possible predictive relationships was formulated. The variable pairs, shown in Table 6.9, unfortunately did not include estimations for the DESIGN and PROGRAM variables. An absence of reliable and consistent figures for these effort indicators, for reasons outlined earlier in this chapter, meant that it would have been impossible to achieve accurate and general predictions for these variables. For each of the prediction pairs shown in Table 6.9 two separate regression procedures were performed. The first allowed a constant term to be included in the regression equation, the second did not. PROGRESS automatically computes the significance of the constant term when it is included—this result, in conjunction with other tests, can then be used to determine which of the two types of equation should be chosen for the prediction. In all seven of the LS tests performed for this sample the constant term was shown to be insignificant. Models that do not include a constant term reflect situations where it is natural to assume that if the predictor variable has a value of zero then the response variable should also have a zero value. For the

current data set this would seem to be quite a reasonable assumption, in that an interactive system that, for example, has no screens, or reads no attributes, would take no days to develop; that is, a system of this type would not exist within the application area of this study.

| Response Variable | Predictor Variable |
|---|---|
| AN_DES | TDSCR |
| | TAU |
| PROG_UT | TAC |
| TOTAL | TRE |
| | TD |
| | TDSCR |
| | TAC |

Table 6.9: Macroanalysis–effort estimation tests

All of the predictor coefficients computed in the non-constant term tests were shown to be significant by the PROGRESS system. This result indicated that the slopes of the computed regression lines were all significantly different from zero. Similarly, the coefficient of determination ($R^2$) values for every equation were shown to be significant using an automatically computed $F$-statistic. This indicated that, in cases where the residuals adhered to the restrictions mentioned in the previous chapter, the predictor variable in each equation did indeed account for the response variable in a significant way. The overall results of these regression tests, including the $R^2$ values, are shown in Table 6.10. As the constant term was found to be insignificant in all of the LS tests the results of these predictions have not been included in the table.

| Response Variable | Predictor Variable | LS | | LMS | | RLS | | |
|---|---|---|---|---|---|---|---|---|
| | | $R^2$ | Res. OK? | $R^2$ | Res. OK? | $R^2$ | Res. OK? | Points Removed |
| AN_DES | TDSCR | 0.903 | U | 0.848 | N | 0.783 | U | 12,13 |
| | TAU | 0.934 | U | 0.940 | Y | 0.966 | Y | 13 |
| PROG_UT | TAC | 0.857 | U | 0.967 | U | 0.959 | U | 7,9,13 |
| TOTAL | TRE | 0.845 | U | 0.934 | N | 0.934 | U | 13 |
| | TD | 0.862 | N | 0.868 | N | 0.909 | N | 9 |
| | TDSCR | 0.794 | N | 0.955 | Y | 0.958 | Y | 12 |
| | TAC | 0.918 | N | 0.962 | U | 0.969 | Y | 11 |

Table 6.10: Macroanalysis–effort regression test results

Based on the information presented in Table 6.10 final predictive equations were chosen for each of the effort variables investigated. Of the two predictions of analysis and design effort (AN_DES), prediction based on the TAU variable was the most

accurate. The three $R^2$ values obtained from the regressions using this variable were higher than those achieved with the equations based on the TDSCR variable. Furthermore it was unclear as to whether the residual plots of the TDSCR models were satisfactory (where the 'RES. OK?' column contains the letter 'U'), whereas those obtained from the TAU models were adequate. Prediction of program and unit test effort (PROG_UT) was only performed in this study with the TAC variable. The results of this estimation were mixed, in that the $R^2$ values were very high but the residual plots were not completely satisfactory. However, given that this was the only predictor for program and unit test effort to be chosen from this data set, continued use of TAC in estimation will show whether it is indeed an accurate and useful predictor. The choice of an estimation model for total development effort (TOTAL) was between the models based on TDSCR and TAC. Both returned very high coefficients of determination, indicating good explanatory ability, and both models produced adequate or good residual plots. Personal preference or ease of extraction may therefore determine which model is more appropriate for an individual project manager. Hence, both models were selected as useful. To summarise the initial findings of the estimation tests the following equations were chosen:

$$AN\_DES = 0.171TAU$$

$$PROG\_UT = 0.080TAC$$

$$TOTAL = 3.842TDSCR$$

or

$$TOTAL = 0.281TAC.$$

The accuracy of the above equations was then assessed through an examination of the residual and error information provided by each model. A summary of this information appears in Table 6.11.

| | AN_DES TAU | PROG_UT TAC | TOTAL TDSCR | TOTAL TAC |
|---|---|---|---|---|
| Number of systems/functions | 12/13 | 10/13 | 12/13 | 12/13 |
| Highest absolute residual | 26.5 | 29.0 | 77.0 | 58.0 |
| Lowest absolute residual | < 0.5 | < 1.0 | 2.0 | 1.5 |
| Largest overestimate | 23.0 | 11.0 | 77.0 | 30.0 |
| Largest underestimate | 26.5 | 29.0 | 35.5 | 58.0 |
| Total residual | –32.5 | –30.5 | +122.0 | –64.0 |
| Average residual | –2.7 | –3.0 | +10.0 | –5.5 |
| Overall residual | –5.0% | –7.5% | +8.0% | –4.0% |
| Absolute error | 129.0 | 80.0 | 336.5 | 283.0 |
| Average absolute error | 10.7 | 8.0 | 28.0 | 23.5 |
| Overall absolute error | 21% | 20% | 22% | 19% |

Table 6.11: System effort estimation residual and error results

This table provides an insight into the adequacy of the models initially selected from the regression tests. The number of systems/functions entry shows the total number of systems or primitive functions with which the selected equation was developed over the number in the original sample. If the numerator is markedly less than the denominator then the general applicability of the equation may be questionable. The highest absolute residual equals the largest absolute difference between the estimated and observed values of the project management indicator over the set of systems or functions when the estimates have been determined using the equation. This equates to the worst estimate achieved by the equation over the sample. Conversely, the lowest absolute residual equates to the best estimate of the equation. High values for both of these figures indicates a lack of accuracy in the model. The largest overestimate is equal to the greatest difference between estimated and observed values from the sample in cases where the estimated figure is greater than the observed figure; the largest underestimate is determined in the same way, but from cases in the sample in which the estimated figure is lower than the observed. Again, smaller values for both of these figures are to be preferred, as this indicates a model that is not consistently producing predictions that are too high or too low.

The total residual is the sum of the differences between the estimated and observed values for all of the systems or functions used in the regression. An optimal total residual would equal zero; that is, the model should overestimate and underestimate to the same degree, preferably by very small amounts in both cases. The average residual of each equation, probably the most useful of the figures in the table, equals the total residual figure divided by the number of systems or functions. The resultant figure is the average amount of overestimation (+) or underestimation (−) produced by the equation, shown as the number of project management indicator units per system/function. One would again prefer this to be as close to zero as possible. The next item in the table, the overall residual, is a percentage representation of the total residual divided by the sum of the observed values. This allows the residual to be considered in relation to the actual number of effort units expended or error units encountered. A large total residual is always undesirable, but it can be put into perspective using the overall residual figure.

The last three figures in the table are concerned only with the amount of residual, not with the direction. That is, all the residuals are considered to be the outcome of incorrect estimates, irrespective of whether they resulted in over- or underestimation. The absolute error is therefore the sum of the absolute values of the residuals produced by the equation. Similarly, the average absolute error is the maximum scope for error of each system or function in the equation sample. Finally, the overall absolute error reflects the degree of inaccuracy attained by an equation (the absolute error) in relation to the total observed units.

One way in which the information provided in Table 6.11 can be used, apart from simply checking the figures for unacceptably high values (in-house limits may be imposed for some of the figures), is in the comparison of two or more estimation models that are being considered for the prediction of the same variable. In the current sample two candidate equations were selected for the prediction of total

development effort—those based on TDSCR and TAC. Although they were previously considered to be of roughly the same merit, based on the relevant $R^2$ values and the residual plots, the information from Table 6.11 enabled more effective discrimination to be carried out. For the two models only the largest underestimate figure was greater under the TAC-based model than it was under the TDSCR equation. The TDSCR-based equation produced an estimate that was on average ten days greater than the observed total development effort for each system, whereas the TAC-based equation underestimated by only five and a half days per system. Moreover, in terms of the total development effort over the complete sample, the TAC model produced an estimate that was 4% less than the actual effort; on the other hand, the TDSCR-based equation provided an estimate that was 8% over the actual figure. Thus the equation based on the TAC variable was selected as the most accurate in the prediction of total development effort. The following set of equations was therefore finally selected:

$$AN\_DES = 0.171TAU$$

$$PROG\_UT = 0.080TAC$$

$$TOTAL = 0.281TAC.$$

### 6.3.2 Sample Two: Macroanalysis—Errors

This sample contained eleven valid data sets. A small number of significant correlations were identified by the initial procedures. Unfortunately all of the significant relationships occurred for the ERRORS variable—no relationships of significant strength were found for the AMEND variable. Thus no further investigation into the classification or prediction of amendments was performed. The relationships that were found for the ERRORS variable are shown in Table 6.12.

| Perspective | C.M. | Specification Indicators |
|---|---|---|
| User Int. | P | TDREP |
| | S | TDREP |
| Data | P | TAU TAC TAM |
| | S | TAC TAM |

Table 6.12: Significant correlations—macroanalysis indicators and errors

The full correlation tables showed no result for the relationship between TMEL and ERRORS. The coefficient for this relationship could not be computed because all but one of the TMEL data points had the same value. This lack of dispersion in the TMEL data points meant that the variable could be disregarded in the rest of the analysis procedures, as its discriminatory power was minimal. Those variables showing significant and valid correlations for both the Spearman and Pearson statistics were then selected for further analysis. These variables are shown in Table 6.13.

| Perspective | Specification Indicators |
|---|---|
| Data | TAC TAM |

Table 6.13: Macroanalysis–errors indicators chosen for further analysis

The TDREP variable was discarded at this stage as it was found to contain a large number of data points with a value of zero. This left only the two data perspective variables, TAC and TAM, for further analysis. An examination of the intercorrelation between these two variables highlighted a very strong relationship. This is hardly a surprise, given that TAM is simply the total of TAC and TAU for a given system. As the TAC variable was available first and was not a composite variable it was chosen as the sole variable to be used for further analysis. The correlation results for this variable are summarised in Table 6.14. The coefficients were significant at the $\alpha = 0.001$ level.

| Specification Indicator | Error Indicator | Pearson Correlation | Spearman Correlation |
|---|---|---|---|
| TAC | ERRORS | 0.9032 | 0.7586 |

Table 6.14: Macroanalysis–errors variable summary

Classification procedures could only proceed after the normality tests had been carried out. A summary of the results is provided in Table 6.15.

| Indicator | Shapiro-Wilks | Lilliefors | Classification Method |
|---|---|---|---|
| TAC | Non-normal | Normal | Both |
| ERRORS | Non-normal | Non-normal | Non-normal |

Table 6.15: Macroanalysis–errors normality tests

The table shows that a mixed result was obtained from the normality tests for the TAC variable. Classification was therefore carried out using both the normal distribution and boxplot distribution methods.

| Spec. Indic. | Error Indic. | Classif. Method | Classif. Correct (%) | Outliers Correct | Excess Outliers |
|---|---|---|---|---|---|
| TAC | ERRORS | Normal | 63.6 | 1/2 | 1 |
| | | Boxplot | 45.4 | 1/2 | 1 |

Table 6.16: Macroanalysis–errors classification results

The results of the classification tests are summarised in Table 6.16. Although a correct classification rate of 63.6% would seem to be quite useful at first, once again

the spread of the data makes any class-based allocation very approximate, with the very large error classes that were generated.

Estimation for this data set contained just the one test, that of ERRORS and TAC, due to the absence of other significant correlations. Again the constant term was found to be insignificant when it was included in the regression formulation, so it was decided to use a non-constant term predictor model. The resulting regression slope was tested by the PROGRESS system and was shown to be significant, as were the $R^2$ values. A summary of this information is shown in Table 6.17.

| Response Variable | Predictor Variable | LS | | LMS | | RLS | | |
|---|---|---|---|---|---|---|---|---|
| | | $R^2$ | Res. OK? | $R^2$ | Res. OK? | $R^2$ | Res. OK? | Points Removed |
| ERRORS | TAC | 0.697 | U | 0.867 | U | 0.919 | Y | 9,10,11 |

Table 6.17: Macroanalysis–errors regression test results

The regression results, as presented in Table 6.17, showed that the reweighted least squares equation provided useful estimates for the number of user acceptance errors (ERRORS). The equation, using the TAC variable, had a significant $R^2$ value of 0.919 and the associated residual plot conformed to the requirements for valid prediction. Thus the following equation was proposed:

$$ERRORS = 0.015TAC.$$

The residual and error information associated with this equation is shown in Table 6.18.

| | ERRORS TAC |
|---|---|
| Number of systems/functions | 8/11 |
| Highest absolute residual | 19 |
| Lowest absolute residual | 2 |
| Largest overestimate | 19 |
| Largest underestimate | 11 |
| Total residual | -7 |
| Average residual | $< -1$ |
| Overall residual | -5.0% |
| Absolute error | 52 |
| Average absolute error | 7 |
| Overall absolute error | 36% |

Table 6.18: System error estimation residual and error results

### 6.3.3 Sample Three: Microanalysis—Effort

Unfortunately only one system was available for this examination of the relationship between low-level specification measures and development effort. The system consisted of eighteen sub-systems from which specification measures were extracted. Total development effort data for each of the sub-systems had also been recorded, so an examination of the relationship between these two sets of data was possible.

Correlation procedures highlighted just a handful of significant relationships between the effort variable (TOT) and the specification measures. Table 6.19 contains a list of those relationships.

| Perspective | C.M. | Specification Indicators |
|---|---|---|
| User Int. | P | SCR |
| | S | |
| Process | P | FO EM |
| | S | |

Table 6.19: Significant correlations—microanalysis indicators and effort

The correlation tables once again included several variable pairs with blank coefficient results. As in the previous sample this is because the variables concerned, NP, PPI, EEP, OOL and MML, all contained data points with exactly the same value for each of the sub-systems. Therefore correlation tests could not be carried out effectively—these variables were consequently removed from the remainder of the procedures due to their lack of discriminatory power.

Variable selection at this stage was normally based on the attainment of significantly high correlation values for both the Spearman and Pearson statistics, as long as the variables concerned did not include a number of tied data points. Unfortunately this meant that all three variables chosen from the initial correlation procedures, as shown in Table 6.19, would have been discarded. Both the SCR and FO variables contained a large number of tied data points when compared to the number of ties in the TOT data—FO took only three values for the eighteen data points and SCR took six, whereas TOT contained twelve different values—and the EM variable returned a Spearman statistic that was only significant at the $\alpha = 0.01$ level. Furthermore the EM variable also contained some tied values. However in the absence of any valid, highly significant Spearman statistics, and given the fact that the TOT data did include some ties, the EM variable was chosen for the classification and estimation tests. The actual correlation coefficients obtained from the tests on the EM-TOT relationship are shown in Table 6.20.

| Specification Indicator | Effort Indicator | Pearson Correlation | Spearman Correlation |
|---|---|---|---|
| EM | TOT | 0.7077 | 0.4882 |

Table 6.20: Microanalysis–effort variable summary

The Shapiro-Wilks and Lilliefors tests for normality were then performed on the EM and TOT data sets. The results of these tests are shown in Table 6.21. The results suggested the use of both boxplot and normal distribution classification tests.

| Indicator | Shapiro-Wilks | Lilliefors | Classification Method |
|---|---|---|---|
| EM | Normal | Normal | Normal |
| TOT | Non-normal | Normal | Both |

Table 6.21: Microanalysis–effort normality tests

| Spec. Indic. | Effort Indic. | Classif. Method | Classif. Correct (%) | Outliers Correct | Excess Outliers |
|---|---|---|---|---|---|
| EM | TOT | Normal | 66.7 | 2/2 | 1 |
| | | Boxplot | 38.8 | 1/3 | 0 |

Table 6.22: Microanalysis–effort classification results

The classification test results, shown in Table 6.22, were encouraging. The procedure that used the EM variable with parameters of the normal distribution correctly classified two thirds of the primitive level functions in terms of their total development effort requirements. The two effort outliers were also identified by this method, but another function that did not turn out to be an effort outlier was incorrectly classified as such by the specification indicator. Overall, however, given the reduced spread of the TOT data, the classification method could be used with some degree of confidence to obtain intervals of effort for the development of future primitive functions based on the number of non-file data elements used or produced by those functions.

Inclusion of a constant term in the estimation tests again proved to be insignificant and resulted in relatively weak explanatory effectiveness. Thus prediction without a constant term was carried out, with greater success. Both the slope coefficients and the coefficients of determination were found to be significant. The relevant results are shown in Table 6.23.

| Response Variable | Predictor Variable | LS | | LMS | | RLS | | |
|---|---|---|---|---|---|---|---|---|
| | | $R^2$ | Res. OK? | $R^2$ | Res. OK? | $R^2$ | Res. OK? | Points Removed |
| TOT | EM | 0.800 | N | 0.934 | Y | 0.961 | Y | 3,6,7,17,18 |

Table 6.23: Microanalysis–effort regression test results

Table 6.23 shows the very high explanatory efficiency of the EM variable in terms of total development effort (TOT), with an optimum $R^2$ value of 0.961 under

the RLS method. Since the residual plots adhered to the requirements of a valid model the RLS estimation equation showed the greatest ability to accurately predict the total development effort requirements of primitive level functions. The relevant equation was therefore:

$$TOT = 0.139EM.$$

Table 6.24 contains the residual and error information relating to the above TOT prediction equation.

| | TOT EM |
|---|---|
| Number of systems/functions | 13/18 |
| Highest absolute residual | 2.7 |
| Lowest absolute residual | < 0.1 |
| Largest overestimate | 1.5 |
| Largest underestimate | 2.7 |
| Total residual | -3.9 |
| Average residual | -0.3 |
| Overall residual | -6.0% |
| Absolute error | 13.4 |
| Average absolute error | 1.0 |
| Overall absolute error | 19% |

Table 6.24: Primitive function effort estimation residual and error results

### 6.3.4 Sample Four: Microanalysis—Errors

The final sample of this study again consisted of just one system, due to the lack of available project management records at the primitive function level for most of the systems investigated in the macroanalysis samples. The one system that was provided for this analysis procedure was made up of twenty-nine low-level functional units, for which system test error data had been recorded. The system could therefore be used to identify possible relationships between primitive function specification measures and the frequency of system test errors applicable to those low level functions.

Unfortunately some of the microanalysis variables were unavailable from the system specification documents—this included all four user interface measures (SCR, REP, DER and DED) and the data element usage variables from the process model set of measures (DEP and DEC). Therefore an investigation of the relationships between these variables and the frequency of system test errors was impossible. Initial correlation tests using the remainder of the variables identified just two significant associations with the system test error data (STERR). These relationships are shown in Table 6.25.

As in the investigation of sample three above, this set of correlations failed to identify any highly significant relationships between the two sets of data using the

Spearman statistic. Moreover, the PPI variable took just four different values for the twenty-nine functions, deeming it unsuitable for further use. This left the IPM variable for possible investigation. An examination of the actual IPM data set revealed that only twelve distinct values were returned by the twenty-nine functions; however a similar examination of the STERR data showed that this set also contained a large number of ties. Therefore the IPM variable was selected for use in subsequent tests. The relevant correlation data for this relationship appears in Table 6.26.

| Perspective | C.M. | Specification Indicators |
|---|---|---|
| Process | P | PPI IPM |
| | S | |

Table 6.25: Significant correlations—microanalysis indicators and errors

| Specification Indicator | Error Indicator | Pearson Correlation | Spearman Correlation |
|---|---|---|---|
| IPM | STERR | 0.6720 | 0.4139 |

Table 6.26: Microanalysis–errors variable summary

Normality tests of the IPM and STERR variables provided the results shown in Table 6.27. As both results indicated non-normal distributions the boxplot-based classification method was chosen as the most appropriate in this case.

| Indicator | Shapiro-Wilks | Lilliefors | Classification Method |
|---|---|---|---|
| IPM | Non-normal | Non-normal | Boxplot |
| STERR | Non-normal | Non-normal | Boxplot |

Table 6.27: Microanalysis–errors normality tests

| Spec. Indic. | Error Indic. | Classif. Method | Classif. Correct (%) | Outliers Correct | Excess Outliers |
|---|---|---|---|---|---|
| IPM | STERR | Boxplot | 48.3 | 3/4 | 2 |

Table 6.28: Microanalysis–errors classification results

The results of the classification test, shown in Table 6.28, revealed only moderate success. Just under half of the functions were placed into the same class for both the IPM and STERR variables. Rather more encouraging was the fact that three of the four error outliers were identified using the specification indicator; however, two further data points were incorrectly identified by this method. In spite of this,

identification of primitive function error outliers using the process model interconnectivity measure (IPM) may prove to be a worthwhile technique to pursue and test further with other systems.

Estimation of system test errors using the IPM variable at first provided similar results to those obtained in previous studies. That is, inclusion of a constant term was found to be of insignificant benefit and the explanatory power of the constant term models was very weak. More useful results were again obtained using the regression procedures that did not include a constant term. Significant values were achieved for both the slope and the explanatory ability of the resultant models.

| Response Variable | Predictor Variable | LS | | LMS | | RLS | | |
|---|---|---|---|---|---|---|---|---|
| | | $R^2$ | Res. OK? | $R^2$ | Res. OK? | $R^2$ | Res. OK? | Points Removed |
| STERR | IPM | 0.603 | Y | 0.627 | Y | 0.582 | Y | 8,9,11 |

Table 6.29: Microanalysis–errors regression test results

The $R^2$ values for the models, shown in Table 6.29, were somewhat weaker than those achieved in the three previous analyses. Moreover, the reweighted least squares (RLS) regression appeared to be less effective than the LS and LMS methods in this case, at least in terms of the explanatory power of the three model types. This may have been due to the fact that the results of the LMS method, which form the basis of an RLS regression, are less effective when residuals are actually normally distributed (Hampel *et al.* [109]), as was the case for this sample. Thus the LMS technique was chosen as the most efficient regression method for this sample. The resultant prediction equation was therefore:

$$STERR = 0.083IPM.$$

This equation produced the residual and error information depicted in Table 6.30.

| | STERR IPM |
|---|---|
| Number of systems/functions | 29/29 |
| Highest absolute residual | 36 |
| Lowest absolute residual | 0 |
| Largest overestimate | 26 |
| Largest underestimate | 36 |
| Total residual | –63 |
| Average residual | –2 |
| Overall residual | –24.0% |
| Absolute error | 181 |
| Average absolute error | 6 |
| Overall absolute error | 70% |

Table 6.30: Primitive function error estimation residual and error results

## 6.4 Discussion of Results

The implications of the results obtained from the statistical examination of the four samples are now considered in isolation before a summary of general observations and recommendations is made at the conclusion of the chapter.

### 6.4.1 Sample One: Macroanalysis—Effort

It was clear even from the correlation results that a strong relationship existed between a number of the macroanalysis specification measures and some of the system effort measures. Significant associations of interest were:

- the number of read operations and total development effort
- the system decomposition structure and total development effort
- the number of distinct system screens and the analysis and design effort
- the number of distinct system screens and total effort
- the number of attributes read by a system and the program and unit test effort
- the number of attributes read by a system and the total development effort
- the number of attributes updated by a system and the analysis and design effort.

The Spearman coefficients for the above relationships also highlighted the very strong relative correspondence of the sets of observations—this was especially so for the relationships involving the number of distinct screens (TDSCR) and the number of attributes read (TAC). These results provided strong support for the subsequent use of the specification variables in the estimation of system development effort requirements.

The results of the classification procedures were, for this sample, rather less useful than they might have been, due to the large spread of data points for many of the variables. Use of just four classes in each test led, in most cases, to extremely large effort intervals. Moreover because of this large data dispersion more applicable results would only have been forthcoming if a very large number of classes had been used, reducing the advantage of simplicity in the classification approach. Thus although all three effort indicators were classed correctly at least 84.6% of the time, the real applicability of these results is not significant.

Similarly the outlier detection procedure, although quite successful, has only limited benefit in this context. In fact, most of the outlier data points from the distributions in this sample were only outliers because the systems from which they were drawn were large (systems ten to thirteen). Little is to be gained from identifying systems that require large amounts of effort to develop simply because they are larger than most of the systems in the sample. Normalisation using a representative variable can often reduce the influence of size on development data; however, when

this approach was tried with the TESDM variable it resulted in significant decreases in the effectiveness of both the classification and outlier identification procedures. Normalisation was therefore abandoned as a solution to the classification difficulties. It is likely that improvements would only have been achieved with the availability of larger samples, leading to a decrease in the class sizes and to a greater degree of effectiveness in determining actual outliers.

The estimation tests, however, were quite successful for this first sample. The explanatory power of each of the three final equations was greater than 0.959, the residual plots all conformed to the requirements of valid predictor models and the overall residuals were all less than 7.5%. In three of the four estimations 92% of the original data points were included, while the fourth estimation included 77% of the original observations. Finally, the equations led to discrepancies of between just 2.7 and 5.5 days per system over the samples included. All of these factors provide support for the accuracy of the equations developed. Furthermore the high degree of inclusion, at levels of 77% and 92% for the four equations, is an encouraging illustration of the general applicability of the equations.

### 6.4.2 Sample Two: Macroanalysis—Errors

Correlation tests produced mixed results for this investigation—significant relationships were found between a number of specification measures and the ERRORS variable, but none were evident for the AMEND data. This suggests that post-delivery amendments were influenced by more than just the function of the systems being considered. It may have also been the result of counting and collection anomalies, that is, amendment figures may have actually included requests for functional enhancements and additions as well as changes to originally specified functionality. These suggestions reflect the fact that as a system comes into operation, users recognise the potential of the system and consequently add to their demands for functionality. Amendment figures may also have been influenced by the system development method employed. If a system had been developed using a prototyping methodology then there should have been, in theory at least, relatively few post-delivery change requests. Thus the absence of any significant relationships for the AMEND variable is not without explanation; however it does indicate that the selection of amendment data as a property largely determined by functionality alone was inappropriate.

Hence the remainder of the examination for this sample was performed only with regard for the occurrence of errors. The final relationship selected from the correlation results was:

- the number of attributes read by a system and the number of functional errors reported during user acceptance testing.

A significant Spearman correlation statistic between the variables also provided evidence of the relative correspondence of the two sets of observations.

Classification again proved to be ineffective because the already small sample contained such a large spread of values. This lack of success was reinforced further

by the low success rate—just 63% of the eleven data points were classified correctly. Outlier identification also proved to be redundant at the systems level—larger systems can generally be expected to contain a proportionately greater number of post-delivery errors. However, a potentially useful outcome of the procedure, although only applicable after development is complete, is that it may enable managers to identify and investigate systems that are error outliers and *not* specification measure outliers, as in system ten of this sample.

The estimation results for this investigation were rather more encouraging. A very strong $R^2$ value was achieved by the final equation, the residual plot was adequate for a valid model, the overall residual was just 5% and, on average, the predictions were out by less than one error per system. It should be noted that the equation only predicts the number of functional errors, not the severity of those errors. An indication of this type, however, should still be useful in the allocation of testing resources and similar tasks. The only drawback to this success was the fact that three systems, numbers nine through eleven, were discarded in the formulation of the equation. These were by far the largest systems in the sample. This suggests that the final equation may only be applicable for systems within a certain size interval. However, the other eight systems still constituted 73% of the original sample—this represents a satisfactory success rate in the estimation of functional errors using the number of attributes read by a system.

### 6.4.3 Sample Three: Microanalysis—Effort

The results of the correlation tests for this sample failed to provide clear evidence of any significant relationships between the primitive function specification measures and total development effort. Subsequent examination of the distribution of the EM variable, however, supported further investigation of the relationship between this data set and the corresponding development effort observations. Although evidence of a linear association did exist in the Pearson statistic, the relative correspondence of the two data sets, confounded somewhat by tied values in both sets, was lower than that obtained in the two previous investigations. This suggested that the EM variable was not an effective *relative* indicator of total effort; however it could still have been *linearly related* to development effort, warranting further examination:

- the number of elements manipulated and the total development effort of primitive functions.

As the analysis of this sample was performed at a much finer level of detail than that of the two earlier analyses the classification procedure proved to be much more effective. The smaller classes enabled two thirds of the functional primitives to be correctly allocated, in terms of their effort requirements, based only on the values of the EM variable. Furthermore, both effort outliers were identified as such by the variable. These results suggest that a project manager could, in 67% of cases, correctly estimate the number of days of effort that would be needed for the development of functional primitives within an interval of, at most, plus or minus two days, based on the number of non-file data elements used and produced by

a process model primitive. They also support the assertion that a manager could correctly identify the primitive functions that would take a greater amount of effort to develop than the majority of functions. This should be of significant assistance in resource and personnel allocation activities.

Prediction proved to be relatively successful. A final $R^2$ value of greater than 0.961 was achieved from a valid estimation model. The predictions were out by just 0.3 days per primitive function, with an overall residual of 6%. However, only thirteen of the eighteen primitive functions were included in the equation after the removal of the other five due to their excessive LMS residuals. This means that the accuracy of the equation applied to just 72% of the original data set. However, this is not necessarily a bad thing. The results showed that primitive functions three, six, seven, seventeen and eighteen were removed. Investigation of these data points showed that four of the five had disproportionately large TOT values in relation to their EM values. Information such as this would enable the manager to further investigate these observations in an attempt to find out why they incurred such high demands on development effort. Therefore estimation of total development effort based on the number of data elements used and produced by process model primitives should continue.

### 6.4.4 Sample Four: Microanalysis—Errors

This investigation turned out to be the weakest of the four. Correlation tests provided only scant evidence of any useful relationships and the coefficients were once again confounded by large numbers of tied data points. The choice of the IPM variable for further examination was based on its significant Pearson statistic, in the absence of any other promising associations:

- process model interconnection and system test errors for primitive functions.

When the normality tests subsequently showed that both variables were non-normally distributed, however, expectations of useful results lessened considerably as the Pearson statistic is not reliable under these circumstances. Classification results returned a correct classification of just over 48%, the lowest result of the four samples. Almost half of the STERR data points were from the lowest distribution class, reflecting the skewed nature of the distribution. Of the four error outliers, three were correctly identified by the IPM classification. Although two further values were incorrectly described as outliers according to the specification variable, this procedure was at least partially successful and may enable managers to pinpoint particularly error-prone primitive functions at the specification stage.

Not unexpectedly, the estimation results obtained from this investigation were also rather disappointing. A maximum $R^2$ of 0.627 was attained from the LMS model, with an acceptable residual pattern. The average residual was an unacceptably high two errors per primitive function and the overall residual was 24%, providing further evidence of a lack of accuracy in the model. There is therefore no evidence in the results of this study to suggest that system test errors may be estimated accurately using specification measures.

## 6.5 Evaluation Summary and Recommendations

The results of the empirical evaluation of the proposed complexity analysis scheme, as presented and discussed in the two previous sections, produced a number of useful findings. The majority of the results confirmed the assumption that specification-based complexity indicators can be used in the effective discrimination of systems and functions in terms of project management consequences. Although the last of the four analyses failed to provide any potentially useful results in terms of practical assistance for project management, the evidence supporting the findings of the other three analyses would appear to be significant.

This degree of significance is reduced somewhat when it is considered that the results of the analysis of sample three, although strong, were derived from the investigation of a single system. It would therefore be inadvisable to extrapolate these results to other projects. However, given the somewhat exploratory nature of this study the results do at least provide first support for the assessment of complexity, in relation to development effort, at the primitive function level. The results obtained from the two macroanalysis-level analyses, however, do provide strong support for system level assessment. The relationships in question were established using data from a number of sites so large differences could have been expected, but the resultant estimation equations were found to be applicable to between 73% and 92% of the original data sets. Furthermore the estimation models all appeared to be valid in terms of residual dispersion requirements. It is therefore envisaged that continued use of the complexity analysis scheme, particularly in an automated collection environment, will provide significant support to project managers.

The statistical procedures chosen were found to be generally appropriate, although the nature of some of the data sets meant that several variables were discarded before analysis and that the statistical methods were not appropriate in every case. For example, correlation procedures were confounded in some cases by the existence of variables with large numbers of ties. Some, such as PPI and NP in sample three, returned only one value for all of the data points. It is now clear that variables such as these, that are likely to include large numbers of tied values, should not be collected, let alone analysed, as their lack of discriminatory power renders them inappropriate for classification and estimation purposes. Similarly, problems were caused by the large dispersion of values evident in some of the variable distributions. This meant that the classification procedures were generally less effective than had been anticipated.

An absence of consistent records relating to analysis, design, program and system test effort meant that these indicators were removed prior to the analysis. Although this was unfortunate, in terms of obtaining results applicable to individual phases of development, removal of the data was certainly more valid than performing statistical analysis on the inconsistent and incomplete data sets. Moreover, it is felt that the composite measures created during the observational work (AN_DES and PROG_UT) were appropriate replacements, and that the estimates obtained for these effort indicators will still be useful in project management.

At the macroanalysis level classification procedures were generally ineffective,

but the estimation methods were successful. The following procedures are therefore recommended:

- estimation of system level effort parameters based on data model measures
  - estimation of analysis and design phase effort, in person-days, using the measure of the number of attributes updated by a system

$$AN\_DES = 0.171TAU$$

  - estimation of programming and unit test effort, in person-days, using the measure of the number of attributes read by a system

$$PROG\_UT = 0.080TAC$$

  - estimation of total development effort, in person-days, using the measure of the number of attributes read by a system

$$TOTAL = 0.281TAC$$

- estimation of user-acceptance phase errors using the measure of the number of attributes read by a system

$$ERRORS = 0.015TAC.$$

As stated several times in this chapter, the results obtained from the microanalysis investigations were based on data extracted from just one system each. Although this restricts the development of generally applicable conclusions, the following recommendations may be useful when examining the results of future studies at this level:

- classification of primitive functions using the EM variable, in order to provide total development effort intervals and to assist in the identification and prediction of effort outlier data points
- estimation of primitive function development effort using the measure of the number of non-file data elements used and produced by a primitive process

$$TOT = 0.139EM.$$

Although some or all of the five equations above are likely to change in the future as more extensive studies are performed, they are at this point recommended as being generally applicable to commercial system development projects undertaken in an extensively automated environment. One of the goals of this study was the development of a functional assessment technique that needed a lesser degree of calibration than most other methods. Given that generally applicable equations have been developed in spite of the widespread differences in products, people and projects encountered in this study, it is hoped that extensive calibration at other sites will indeed no longer be necessary.

# Chapter 7

# Conclusions and Recommendations

## 7.1 Summary and Conclusions

This study set out to develop and validate a specification-based functional complexity analysis scheme applicable to interactive commercial systems. To this end, previously proposed complexity assessment methods were examined so that a basis for improvement could first be established. The subsequently proposed analysis scheme was developed as a direct response to the failings of previous methods, addressing issues such as subjectivity and environment dependence. The scheme was then tested using data collected from sixteen projects developed at ten different sites. Strong evidence of significant, useful relationships was provided using robust statistical analysis methods, confirming the assertion that specification measures were related to project management data. Recommendations for project management were therefore made, based on the results obtained from the analysis.

Chapter 1 showed that, in terms of new project parameter estimation, traditional approaches to complexity assessment have little to offer. Results of the lexical and topological techniques are clearly not available at the conceptual development stage of a project, when managers need to justify cost and effort requirements. Similarly, most structural methods can only be applied after a significant amount of effort, and therefore expense, has been invested in a project. This is clearly undesirable under increasingly tight economic constraints. Support for functional assessment methods is therefore strong—their development and use, however, is not yet widespread. Functional software specification methods were discussed in Chapter 2, in order to provide a basis for the development of appropriate functional assessment techniques. To this end, the impact of increasing software development automation was also considered. These discussions revealed strong support for the use of data-centred specification notations as a basis for functional complexity assessment in an automated environment. With this in mind, currently proposed functional analysis techniques were examined in Chapter 3. This examination highlighted a number of areas of concern, particularly in relation to the subjective nature of many of the techniques and to the extensive dependence of the methods on personnel and

environment considerations.

The extensive literary support for functional assessment, coupled with the need for more objective and comprehensive techniques, provided direct motivation for the development of the specification-based analysis scheme proposed in Chapter 4. Use of the GQM and Classification paradigms enabled the structured selection of appropriate specification and project management measures. This resulted in the development of a scheme that was intended to achieve two main aims: (i) to overcome the problems of previously proposed techniques; and (ii) to satisfy the empirical objectives of the study. Achievement of these objectives, however, was preceded by a consideration of the theoretical validity of the scheme. Although the proposal was shown to be valid according to most of the recent discussions on theoretical criteria, some deviations were noted and justification for these differences was provided. The empirical requirements of the study were then discussed, with particular emphasis being placed on the need to use robust statistical analysis in order to obtain valid results.

Analysis of the collected data sets provided evidence to suggest that the scheme had succeeded in achieving the two aims mentioned above. It had certainly satisfied the requirements for an early, objective, comprehensive, independent and validated approach. The products of the statistical analysis also supported the achievement of the empirical objectives. Although mixed results were obtained from some of the empirical analyses, the outcome of most of the procedures was generally useful, providing strong evidence of relationships between specification measures and project management data. Recommendations based on this evidence were therefore provided for the discrimination and estimation of both development effort and post-delivery error occurrence. Given that this analysis was the first empirical validation of its type, the study has been somewhat exploratory in nature. It is hoped, however, that the accuracy and applicability of the results will provide a sound basis for future observational studies.

Thus the overall goal of this study, the development and validation of an appropriate complexity analysis scheme, was achieved. Similarly the research objectives stated in Chapter 1 were also satisfied—complexity assessment failings were identified, a new strategy was developed, relationships between complexity indicators and project management data were established, classification based on these relationships was performed and predictive equations for project management parameters were developed. The accuracy and general applicability of the equations provides evidence supporting the assertion that complexity analysis could be performed without consideration of organisational or personnel factors. The approach is still dependent on technology, in that it assumes the use of structured methods in an automated environment. This is not seen as a major constraint, however, as these methods and tools are widely used in the commercial software development domain.

A detailed examination of the analysis results reveals that data model variables were selected as those most closely related to effort and error data at the macroanalysis level. This may be taken to suggest that the data model is the most appropriate of the specification representations, in terms of providing a basis for complexity assessment. It should be remembered, however, that process model data was not

available for these analyses. This conclusion must therefore remain unconfirmed until further studies can be performed. In the microanalysis investigations process model variables were found to have the closest relationship with project management data. However the conclusions reached based on these procedures cannot be generalised, due to their basis in single system samples.

The results of the system-level analyses were particularly encouraging in that relatively useful predictive equations were developed from data sets collected from different sites employing different people, products and procedures. One factor that may have contributed to this outcome is the fact that the sites were all relatively mature and committed CASE/4GL product users, providing some support for the assertion that automation is a leveller of sorts. The extensive use of automation therefore enabled objective, quantitative assessment to be performed, leading to the development of simple but relatively accurate estimation equations.

It was suggested in Chapter 1 that user satisfaction was indirectly influenced by software complexity. Although this dissertation has attempted to develop methods that enable some sort of control to be established over complexity, user satisfaction may still remain elusive. The goal of estimation studies such as this one must be to assist in the delivery of a quality software product on time according to estimates. If the product no longer matches the requirements of the user, because these requirements have changed but have not been incorporated into the specification, then this is unfortunate and it is a reflection of the hazards of software development in a dynamic environment. It is not, however, a failure of the assessment and estimation processes.

The findings of this study should therefore provide assistance to researchers and practitioners alike. The results and discussions of the previous chapter should form the basis for effective, objective and early discrimination and estimation of development effort and post-delivery errors in the commercial software development environment. Use of the recommended procedures by project managers should enable them to more effectively control the influence that functional complexity has on their development projects.

## 7.2 Recommendations for Research

The summary at the end of the previous chapter included a number of suggestions for appropriate and effective project management procedures, based on the analysis results of this study. Ongoing use of these procedures and equations should enable managers to obtain frequent and accurate indications of effort requirements and error occurrence. The equations, however, are only applicable to small or medium sized interactive commercial systems developed with extensive automated assistance. Given that the data from some of the larger systems was discarded during the analyses before the equations were developed, it may be that other equations will be more appropriate for the prediction of effort requirements and error occurrence for larger systems. Further research and analysis of larger systems will provide evidence to support or refute this remark. More extensive analysis of systems developed with

other automated tools is also needed. This will produce more comprehensive data sets, enabling more general use of the resultant procedures and equations. Extensive investigations of systems specified using process models is also required, so that the consideration of process model measures, which had to be abandoned in this study because of a lack of relevant data, can still be performed.

Similarly, reinforcement of the results obtained from small and medium sized systems will be forthcoming as larger samples become available for analysis and as collection becomes increasingly automated within development tools. It is envisaged that the current scheme will be incorporated into a CASE tool in the near future. This will have two advantages over the current study: firstly, it will enable more objective, non-intrusive, less error-prone collection of the data to be carried out without the need for time-consuming manual collection; secondly, it will mean that analysis and prediction may be performed in the background of development as an integral part of a project. Tate [240] and Tate and Verner [243] also suggest that on-workbench data, relating to development effort, will soon be collected automatically within CASE environments. Collection of project management data will therefore also be more precise and cost-effective. All of these factors will encourage continuing refinement of the equations, providing relevant feedback to managers whenever required.

Although the proposal developed in this study was comprehensive in terms of the commercial software specification representations examined, consideration of a number of other notations may prove to be useful in future analyses. Recent studies investigating the combination of petri nets with data flow diagrams, to add rigour to process development (Benwell *et al.* [17]; Tse and Pong [249]; Lee and Tan [157]), highlight a combined environment that may be a more appropriate foundation for process model assessment. Similarly, object-oriented development, which is steadily becoming more established in the commercial development domain, may also provide a useful basis for analysis. The combination of data and process into objects reduces the distinction between the two at the code level. At the conceptual specification level, however, this distinction is still present (Macdonald [167]). However it is augmented by the specification of events so this overall representation may need to be considered in an object-oriented environment.

A similar approach within structured development methods would be to include the consideration of entity life histories, which describe the states that entities assume during system operation, or event lists, which describe the triggers that cause data to begin or stop flowing (Keuffel [141]; Symons [236]; Robinson [204]). Macdonald [167] also suggests that control conditions and state transitions could be important. It is currently unclear as to whether these specification methods are widely used in the commercial software development domain. If, however, they do become standard approaches then their consideration should be encouraged.

The possible incorporation of formal methods into commercial software development would also provide further scope for analysis. An environment of this type would enable straightforward determination and specification of requirements using traditional structured methods, but would also enable validation of consistency and completeness to be performed (Babin *et al.* [7]; Fraser *et al.* [90]), resulting in more

rigorous development. Bishop and Lehman [21] and Fenton [79], however, state that formal methods are difficult to use, particularly for the non-mathematically inclined. They suggest that CASE support is necessary if they are to become widely used. Furthermore, Tao and Kung [239] state that even with formal methods it is still difficult to show that commercial system requirements have actually been met, because the language used to describe businesses and organisational processes is different from current formal specification languages. For example, business activities have deadlines and are started or stopped by time-dependent triggers; temporality, however, is difficult to express in formal specifications (Denning [71]). In spite of these problems, Forte and Norman [89] state that the attempts to incorporate formal methods within commercial development techniques should be pursued as their use will introduce a greater degree of rigour into the development process. If and when this integration does occur, further functional complexity analysis will become possible.

Finally an issue of increasing importance within the software development community is that of reuse, that is, the incorporation of existing software components into new systems. This study has not considered reuse in its assessment, as all of the systems used for validation of the proposal were developed from new requirements. However, the current approach would not be appropriate in projects where previously developed components are to be used. Clearly reused components have already progressed through the analysis-design-construction-test phases, so effort and error predictions for these components using the current equations would be incorrect. Some consideration of the complexity of the reused component specifications, in terms of the contribution that they make to the overall specification measures, should therefore be undertaken.

Until software development in the commercial environment becomes a totally automated procedure, functional complexity will continue to be an important influence on the progress and outcomes of the development process. Continually rising development costs, coupled with more and more demands for increasingly complicated systems, will encourage extensive research into quantifiable assessment/estimation methods and into development automation. It is therefore hoped that this study, which has empirically considered the interaction of these factors, will form the basis for continued research in this area.

# References

[1] Alavi, M. and Wetherbe, J.C., "Mixing Prototyping and Data Modeling For Information System Design", *IEEE Software*, May 1991, pp. 86-91.

[2] Albrecht, A.J., "Measuring Application Development Productivity", *Proceedings IBM GUIDE/SHARE Applications Development Symposium*, California, 1979.

[3] Albrecht, A.J., *Open Letter to the Secretary of the International Function Point User Group*. IFPUG Memorandum, June 1988.

[4] Albrecht, A.J. and Gaffney, J.E. Jr, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering* 9 (6), November 1983, pp. 639-648.

[5] Allen, R.E. (ed.), *The Pocket Oxford Dictionary of Current English*. Oxford University Press (7th ed.), Oxford, 1984.

[6] Arthur, L.J., *Measuring Programmer Productivity And Software Quality*. John Wiley & Sons, New York, 1985.

[7] Babin, G., Lustman, F. and Shoval, P., "Specification and Design of Transactions in Information Systems: A Formal Approach", *IEEE Transactions on Software Engineering* 17 (8), August 1991, pp. 814-829.

[8] Baker, A.L., Bieman, J.M., Fenton, N.E., Gustafson, D.A., Melton, A. and Whitty, R., "A Philosophy for Software Measurement", *Journal of Systems and Software* 12, 1990, pp. 277-281.

[9] Banker, R.D. and Kauffman, R.J., "Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study", *MIS Quarterly*, September 1991, pp. 375-401.

[10] Basili, V.R. and Rombach, H.D., "The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Transactions on Software Engineering* 14 (6), June 1988, pp. 758-773.

[11] Basili, V.R. and Weiss, D.M., "A Methodology for Collecting Valid Software

Engineering Data", *IEEE Transactions on Software Engineering* 10 (6), November 1984, pp. 728-738.

[12] Bastani, F.B., "An Approach To Measuring Program Complexity", *Proceedings COMPSAC '83*, Chicago IL, 1983, pp. 1-8.

[13] Batini, C., Lenzerini, M. and Navathe, S.B., "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys* 18 (4), December 1986, pp. 323-364.

[14] Baxter, I.D., "Design Maintenance Systems", *Communications of the ACM* 35 (4), April 1992, pp. 73-89.

[15] Beane, J., Giddings, N. and Silverman, J., "Quantifying Software Designs", *Proceedings 7th International Conference on Software Engineering*, Orlando FL, 1984, pp. 314-322.

[16] Belson, M. and Devonald, S., "Using Design Templates with IEW", *Proceedings KnowledgeWare International User Conference*, Atlanta GA, April 1991, pp. 571-579.

[17] Benwell, G.L., Firns, P.G. and Sallis, P.J., "Deriving Semantic Data Models from Structured Process Descriptions of Reality", *Journal of Information Technology* 6 (1), March 1991, pp. 15-25.

[18] Berns, G.M., "Assessing Software Maintainability", *Communications of the ACM* 27 (1), January 1984, pp. 14-23.

[19] Bersoff, E.H., Henderson, V.D. and Siegel, S.G., *Software Configuration Management - An Investment in Product Integrity*. Prentice-Hall, Englewood Cliffs NJ, 1980.

[20] Bhide, S., "Generalized Software Process-Integrated Metrics Framework", *Journal of Systems and Software* 12, 1990, pp. 249-254.

[21] Bishop, R. and Lehman, M.M., "A View of Software Quality", *Proceedings IEE Colloquium on Designing Quality into Software Based Systems*, (PGC1, No. 1991/151), London, October 1991, pp. 1/1-1/3.

[22] Blaha, M.R., Premerlani, W.J. and Rumbaugh, J.E., "Relational Database Design Using An Object-Oriented Methodology", *Communications of the ACM* 31 (4), April 1988, pp. 414-427.

[23] Blaine, J.D. and Kemmerer, R.A., "Complexity Measures for Assembly Language Programs", *Journal of Systems and Software* 5, August 1985, pp. 229-245.

[24] Bobbie, P.O., "Productivity Through Automated Tools", *ACM SIGSoft Software Engineering Notes* 12 (2), April 1987, pp. 30-31.

[25] Boehm, B.W., *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs NJ, 1981.

[26] Boehm, B.W., Gray, T.E. and Seewaldt, T., "Prototyping Versus Specifying: A Multiproject Experiment", *IEEE Transactions on Software Engineering* 10 (3), May 1984, pp. 290-302.

[27] Boehm, B.W. and Papaccio, P.N., "Understanding and Controlling Software Costs", *IEEE Transactions on Software Engineering* 14 (10), October 1988, pp. 1462-1477.

[28] Bollmann, P. and Zuse, H., "An Axiomatic Approach To Software Complexity Measures", *Proceedings 3rd Symposium on Empirical Foundations of Information and Software Science*, Roskilde, 1985, pp. 13-20.

[29] Bowker, P., "The Challenge for Software Development", in Gillies, A. (ed.), *Case Studies in Software Engineering.* Salford University Business Services Ltd, UK, March 1991, pp. 9-18.

[30] Bowman, B.J. and Newman, W.A., "Software Metrics as a Programming Training Tool", *Journal of Systems and Software* 13, 1990, pp. 139-147.

[31] British Gas, *Bang Metric Analysis.* Document Num. 000763, Process Support, British Gas plc, Dorking, June 1991.

[32] Brooks, F.P. Jr, "No Silver Bullet – Essence and Accidents of Software Engineering", *Computer* 20 (4), April 1987, pp. 10-19.

[33] Brown, A.W. and McDermid, J.A., "Learning from IPSE's Mistakes", *IEEE Software*, March 1992, pp. 23-28.

[34] Brown, D.W., Carson, C.D., Montgomery, W.A. and Zislis, P.M., "Software Specification and Prototyping Technologies", *AT&T Technical Journal*, July/August 1988, pp. 33-45.

[35] Buckler, G., "Speeding Up the Programming Process", *Computing Canada* 14 (11), 26 May 1988, pp. 26-27.

[36] Burkhard, D.L. and Jenster, P.V., "Applications of Computer-Aided Software Engineering Tools: Survey of Current and Prospective Users", *ACM SIGBDP Data Base*, Fall 1989, pp. 28-37.

[37] Bush, M.E. and Fenton, N.E., "Software Measurement: A Conceptual Framework", *Journal of Systems and Software* 12, 1990, pp. 223-231.

[38] Bushell, C.J., "The Strengths Of Data Modelling", *Proceedings 18th CAE Computer Conference*, Australia, 1987, pp. 105-117.

[39] Card, D.N., Page, G.T and McGarry, F.E., "Criteria for Software Modularization", *Proceedings 8th International Conference on Software Engineering*, London, 1985, pp. 372-377.

[40] Case, A.F. Jr, *Information Systems Development: Principles of Computer-Aided Software Engineering.* Prentice-Hall, Englewood Cliffs NJ, 1986.

[41] Chapin, N., "A Measure of Software Complexity", *Proceedings 1979 National Computer Conference*, New York, 1979, pp. 995-1002.

[42] Chen, E.T., "Program Complexity and Programmer Productivity", *IEEE Transactions on Software Engineering* 4 (3), May 1978, pp. 187-194.

[43] Chen, M. and Norman, R.J., "A Framework for Integrated CASE", *IEEE Software*, March 1992, pp. 18-22.

[44] Chen, M., Nunamaker, J.F. Jr and Weber, E.S., "Computer-Aided Software Engineering: Present Status and Future Directions", *ACM SIGBDP Data Base*, Spring 1989, pp. 7-13.

[45] Chen, P.P., "The Entity Relationship Model: Towards a Unified View of Data", *ACM Transactions on Database Systems* 1 (1), March 1976, pp. 9-36.

[46] Chen, P.P. (ed.), *Entity-Relationship Approach - The Use of E-R Concept in Knowledge Representation.* IEEE Computer Society Press, Washington, 1985.

[47] Cherniavsky, J.C. and Smith, C.H., "On Weyuker's Axioms for Software Complexity Measures", *IEEE Transactions on Software Engineering* 17 (6), June 1991, pp. 636-638.

[48] Chikofsky, E.J. and Rubenstein, B.L., "CASE: Reliability Engineering for Information Systems", *IEEE Software*, March 1988, pp. 11-16.

[49] Choong, L.S. and Churcher, N., "Macaw: An Entity-Relationship Diagrammer for the Macintosh", *New Zealand Journal of Computing* 1 (1), April 1989, pp. 66-72.

[50] CIS, *CASE Project - Report on Computer Aided Software Engineering in New Zealand.* Case Study Report, Center for Information Science, University of Auck-

land, May 1989.

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

[52] Colligan, I.C. and Nevill, D.G., "Software Metrics for Planning and Control", *Information Management*, Winter 1988, pp. 13-18.

[53] Compton, B.T. and Withrow, C., "Prediction and Control of ADA Software Defects", *Journal of Systems and Software* 12, 1990, pp. 199-207.

[54] Conte, S.D., Dunsmore, H.E. and Shen, V.Y., *Software Engineering Metrics and Models*. Benjamin/Cummings Publishing, Menlo Park CA, 1986.

[55] Côté, V., Bourque, P., Oligny, S. and Rivard, N., "Software Metrics: An Overview of Recent Results", *Journal of Systems and Software* 8, 1988, pp. 121-131.

[56] Coulter, N.S., "Software Science and Cognitive Psychology", *IEEE Transactions on Software Engineering* 9 (2), March 1983, pp. 166-171.

[57] Coupal, D. and Robillard, P.N., "Factor Analysis of Source Code Metrics", *Journal of Systems and Software* 12, 1990, pp. 263-269.

[58] Crozier, M., Glass, D., Hughes, J.G., Johnston, W. and McChesney, I., "Critical Analysis of Tools for Computer-Aided Software Engineering", *Information and Software Technology* 31 (9), November 1989, pp. 486-496.

[59] Curtis, B., "The Measurement of Software Quality and Complexity", in Perlis, A.J., Sayward, F.G. and Shaw, M. (eds.), *Software Metrics*. MIT Press, Massachusetts, 1981, pp. 203-224.

[60] Curtis, B., "Software Metrics: Guest Editor's Introduction", *IEEE Transactions on Software Engineering* 9 (6), November 1983, pp. 637-638.

[61] Curtis, B., Sheppard, S.B., Milliman, P., Borst, M.A. and Love, T., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", *IEEE Transactions on Software Engineering* 5 (2), March 1979, pp. 96-104.

[62] Daniel, W.W., *Applied Nonparametric Statistics*. PWS-Kent Publishing Company, Boston MA, 1990.

[63] Date, C.J., *An Introduction to Database Systems*. Addison-Wesley, New York, 1986.

[64] Davis, J.S., "Chunks: A Basis For Complexity Measurement", *Information Processing & Management* 20 (1-2), 1984, pp. 119-127.

[65] Davis, J.S. and LeBlanc, R.J., "A Study of the Applicability of Complexity Measures", *IEEE Transactions on Software Engineering* 14 (9), September 1988, pp. 1366-1372.

[66] Dawson, K.S. and Purgailis Parker, L.M., "From Entity-Relationship Diagrams to Fourth Normal Form: A Pictorial Aid to Analysis", *The Computer Journal* 31 (3), 1988, pp. 258-268.

[67] DeMarco, T., *Structured Analysis and System Specification.* Prentice-Hall, Englewood Cliffs NJ, 1978.

[68] DeMarco, T., *Controlling Software Projects.* Yourdon, New York, 1982.

[69] DeMarco, T., "An Algorithm For Sizing Software Products", *ACM SIGMetrics Performance Evaluation Review* 12 (2), 1984, pp. 13-22.

[70] Demurjian, S.A. and Hsiao, D.K., "Towards a Better Understanding of Data Models Through the Multilingual Database System", *IEEE Transactions on Software Engineering* 14 (7), July 1988, pp. 946-958.

[71] Denning, P.J., "Editorial - What is Software Quality?", *Communications of the ACM* 35 (1), January 1992, pp. 13-15.

[72] Dunsmore, H.E., "Software Metrics: An Overview of an Evolving Methodology", *Information Processing & Management* 20 (1-2), 1984, pp. 183-192.

[73] Eglington, D., "Cost-Effective Computer System Implementation in Medium Sized Companies", in Gillies, A. (ed.), *Case Studies in Software Engineering.* Salford University Business Services Ltd, UK, March 1991, pp. 56-59.

[74] Eisenbach, S., McLoughlin, L. and Sadler, C., "Data-Flow Design as a Visual Programming Language", *Proceedings 5th International Workshop on Software Specification and Design/ACM SIGSoft Software Engineering Notes* 14 (3), May 1989, pp. 281-283.

[75] Ejiogu, L.O., "A Simple Measure Of Software Complexity", *ACM SIGMetrics Performance Evaluation Review* 13 (1), June 1985, pp. 33-47.

[76] Evangelist, W.M., "Software Complexity Metric Sensitivity to Program Structuring Rules", *Journal of Systems and Software* 3, 1983, pp. 231-243.

[77] Evangelist, W.M., "An Analysis Of Control Flow Complexity", *Proceedings*

*COMPSAC '84*, Chicago IL, 1984, pp. 388-396.

[78] Factor, R.M. and Smith, W.B., "A Discipline for Improving Software Productivity", *AT&T Technical Journal*, July/August 1988, pp. 2-9.

[79] Fenton, N.E., "Software Metrics: Theory, Tools and Validation", *Software Engineering Journal*, January 1990, pp. 65-78.

[80] Fenton, N.E. and Kaposi, A.A., "Metrics and Software Structure", *Information and Software Technology* 29 (6), July/August 1987, pp. 301-320.

[81] Fenton, N.E. and Melton, A., "Deriving Structurally Based Software Measures", *Journal of Systems and Software* 12, 1990, pp. 177-187.

[82] Ferg, S., "Modelling the Time Dimension in an Entity-Relationship Diagram", in Chen, P.P. (ed.), *Entity-Relationship Approach - The Use of ER Concept in Knowledge Representation.* IEEE Computer Society Press, Washington, 1985, pp. 280-286.

[83] Fetzer, J.H., "Program Verification: The Very Idea", *Communications of the ACM* 31 (9), September 1988, pp. 1048-1063.

[84] Finkelstein, C., "Information Engineering", *Computerworld*, May-June 1981.

[85] Finkelstein, C., *An Introduction to Information Engineering.* Addison-Wesley, New York, 1989.

[86] Firns, P.G., "Determining a Useful Balance Between Understandability and Rigour in Data Modelling", *New Zealand Journal of Computing* 2 (1), December 1990, pp. 13-21.

[87] Firns, P.G., "Entity Relationship Modelling in GIS Design: An Efficacious Approach or an Exercise in Futility?", *Proceedings 18th Australasian Conference in Urban and Regional Planning Information Systems*, Australia, 1990.

[88] Fisher, D. and Betteridge, R., *Forecasting ASD Resource Requirements for New Elements.* Internal Report, Inland Revenue IT Division M3/T3, Telford, 14 December 1987.

[89] Forte, G. and Norman, R.J., "A Self-Assessment by the Software Engineering Community", *Communications of the ACM* 35 (4), April 1992, pp. 28-32.

[90] Fraser, M.D., Kumar, K. and Vaishnavi, V.K., "Informal and Formal Requirements Specification Languages: Bridging the Gap", *IEEE Transactions on Software Engineering* 17 (5), May 1991, pp. 454-466.

[91] Freeman, P., "Mapping Specifications to Formalisms: Some Problems with Traditional Approaches to Capturing Specifications", in Chen, P.P. (ed.), *Entity-Relationship Approach - The Use of ER Concept in Knowledge Representation.* IEEE Computer Society Press, Washington, 1985, pp. 100.

[92] Gaffney, J.E. Jr, Goldberg, R. and Misek-Falkoff, L.D., "Score82 Summary", *ACM SIGMetrics Performance Evaluation Review* 12 (4), 1984-85, pp. 4-9.

[93] Gane, C. and Sarson, T., *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, Englewood Cliffs NJ, 1979.

[94] Gavurin, S.L., "Where Does Prototyping Fit In IS Development?", *Journal of Systems Management*, February 1991, pp. 13-17.

[95] Gerritsen, R., Morgan, H. and Zisman, M., "On Some Metrics for Databases or What is a Very Large Database?", *ACM SIGMOD Record* 9 (1), June 1977, pp. 50-74.

[96] Gibson, V.R. and Senn, J.A., "System Structure and Software Maintenance Performance", *Communications of the ACM* 32 (3), March 1989, pp. 347-358.

[97] Glass, R.L., "Editor's Corner - 4GLs and CASE: What's the Payoff?", *Journal of Systems and Software* 14, 1991, pp. 131-132.

[98] Godwin, A.N., Gore, M.B. and Salt, D.W., "A Comparison of JSD and DFD as Descriptive Tools", *The Computer Journal* 32 (3), 1989, pp. 202-211.

[99] Goering, R., "Design Tools Advance to Keep Pace With System Complexity", *Computer Design*, December 1987, pp. 103-119.

[100] Gordon Group, *Before You Leap - A Software Cost Model.* Product User Manual, Gordon Group, San Jose CA, 1987.

[101] Grady, R.B., "Work-Product Analysis: The Philosopher's Stone of Software?", *IEEE Software*, March 1990, pp. 26-34.

[102] Gray, R.H.M., Carey, B.N., McGlynn, N.A. and Pengelly, A.D., "Design Metrics for Database Systems", *BT Technology Journal* 9 (4), October 1991, pp. 69-79.

[103] Gremillion, L.L., "Determinants of Program Repair Maintenance Requirements", *Communications of the ACM* 27 (8), August 1984, pp. 826-832.

[104] Grupe, F.H. and Clevenger, D.F., "Using Function Point Analysis as a Software Development Tool", *Journal of Systems Management*, December 1991, pp.

23-26.

[105] Haddley, N. and Sommerville, I., "Integrated Support for Systems Design", *Software Engineering Journal*, November 1990, pp. 331-338.

[106] Hall, N.R. and Preiser, S., "Combined Network Complexity Measures", *IBM Journal of Research and Development* 28 (1), January 1984, pp. 15-27.

[107] Halstead, M.H., *Elements of Software Science*. Elsevier North-Holland, New York, 1977.

[108] Hamer, P.G. and Frewin, G.D., "M.H. Halstead's Software Science - A Critical Examination", *Proceedings 6th International Conference on Software Engineering*, Tokyo, 1982, pp. 197-206.

[109] Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J. and Stahel, W.A., *Robust Statistics*. John Wiley & Sons, New York, 1986.

[110] Han, W.-T., Choe, Y.-C. and Park, Y.-J., "Software Metrics Using Operand Type", *Proceedings TENCON '87*, Seoul, 1987, pp. 1212-1215.

[111] Harel, D., "Biting the Silver Bullet - Toward a Brighter Future for System Development", *Computer*, January 1992, pp. 8-20.

[112] Harel, E.C. and McLean, E.R., "The Effects of Using a Nonprocedural Computer Language on Programmer Productivity", *MIS Quarterly*, June 1985, pp. 109-120.

[113] Harrison, W.A., "Software Complexity Metrics: A Bibliography and Category Index", *ACM SIGPLan Notices* 19 (2), February 1984, pp. 17-27.

[114] Harrison, W.A. and Cook, C., "A Micro/Macro Measure of Software Complexity", *Journal of Systems and Software* 7, 1987, pp. 213-219.

[115] Harrison, W.A. and Magel, K.I., "A Complexity Measure Based On Nesting Level", *ACM SIGPLan Notices* 16 (3), March 1981, pp. 63-74.

[116] Harwood, K., "On Prototyping and the Role of the Software Engineer", *ACM SIGSoft Software Engineering Notes* 12 (4), October 1987, pp. 34.

[117] Hawryszkiewycz, I.T., *Introduction to Systems Analysis and Design*. Prentice Hall Australia, Sydney, 1988.

[118] Henry, S. and Kafura, D., "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering* 7 (5), September 1981, pp. 510-

518.

[119] Henry, S. and Lewis, J., "Integrating Metrics into a Large-Scale Software Development Environment", *Journal of Systems and Software* 13, 1990, pp. 89-95.

[120] Hoaglin, D.C., Mosteller, F. and Tukey, J.W., *Understanding Exploratory Data Analysis.* John Wiley & Sons, New York, 1983.

[121] Horch, J.W., "Productivity Through Quality", *Proceedings 12th New Zealand Computer Conference*, Dunedin, 1991, pp. 1-4.

[122] Hsu, C., "Structured Database Systems Analysis and Design Through Entity Relationship Approach", in Chen, P.P. (ed.), *Entity-Relationship Approach - The Use of ER Concept in Knowledge Representation.* IEEE Computer Society Press, Washington, 1985, pp. 56-63.

[123] IE, *IE-Metrics Knowledge Base.* James Martin & Co., Reston VA, November 1989.

[124] Ince, D.C., "The Influence of System Design Complexity Research on the Design of Module Interconnection Languages", *ACM SIGPLan Notices* 20 (10), October 1985, pp. 36-43.

[125] Ince, D.C. and Shepperd, M., "System Design Metrics: A Review And Perspective", *Proceedings 2nd IEE/BCS Conference on Software Engineering*, London, 1988, pp. 23-27.

[126] Ivan, I., Arhire, R. and Macesanu, M., "Programs Complexity: Comparative Analysis, Hierarchy, Classification", *ACM SIGPLan Notices* 22 (4), April 1987, pp. 94-102.

[127] Iyengar, S.S., Bastani, F.B. and Fuller, J.W., "An Experimental Study Of The Logical Complexity Of Data Structures", *Proceedings 2nd Symposium on Empirical Foundations of Information and Software Science*, Atlanta GA, 1985, pp. 225-239.

[128] Jackson, M., *Principles of Program Design.* Academic Press, New York, 1975.

[129] Jackson, M., "Software Engineering for Business DP: Looking Back and Looking Forward", *Proceedings 11th International Conference on Software Engineering*, Pittsburgh PA, 1989, pp. 135.

[130] Jarke, M., "Strategies for Integrating CASE Environments", *IEEE Software*, March 1992, pp. 54-61.

[131] Jayaprakash, S., Lakshmanan, K.B. and Sinha, P.K., "MEBOW: A Compre-

hensive Measure Of Control Flow Complexity", *Proceedings COMPSAC '87*, 1987, pp. 238-244.

[132] Jones, R., "A Quantum Leap in Languages: Major Gains in Using 4GLs", *Computerworld New Zealand*, 31 October 1988, pp. 18-19.

[133] Jones, T.C., "Why Choose CASE?", *BYTE*, December 1989, pp. 80IS3-10.

[134] Kafura, D. and Canning, J., "A Validation of Software Metrics Using Many Metrics and Two Resources", *Proceedings 8th International Conference on Software Engineering*, London, 1985, pp. 378-385.

[135] Karimi, J. and Konsynski, B.R., "An Automated Software Design Assistant", *IEEE Transactions on Software Engineering* 14 (2), February 1988, pp. 194-210.

[136] Kearney, J.K., Sedlmeyer, R.L., Thompson, W.B., Gray, M.A. and Adler, M.A., "Software Complexity Measurement", *Communications of the ACM* 29 (11), November 1986, pp. 1044-1050.

[137] Kerr, J.M., "The Information Engineering Paradigm", *Journal of Systems Management*, April 1991, pp. 28-35.

[138] Keuffel, W., "CASE for the Rest of Us", *Computer Language (USA)*, January 1991, pp. 25-29.

[139] Keuffel, W., "Data Modeling in Context", *Computer Language (USA)*, March 1991, pp. 27-33.

[140] Keuffel, W., "Exploring ERD Tools", *Computer Language (USA)*, April 1991, pp. 27-35.

[141] Keuffel, W., "Event Lists and Essential Models", *Computer Language (USA)*, May 1991, pp. 29-34.

[142] Keuffel, W., "Building Essential DFDs", *Computer Language (USA)*, June 1991, pp. 29-41.

[143] Keuffel, W., "The Structured Specification Data Dictionary", *Computer Language (USA)*, July 1991, pp. 29-34.

[144] Keuffel, W., "Transformation Strategies", *Computer Language (USA)*, October 1991, pp. 35-42.

[145] Kilov, H., "Conventional and Convenient in Entity-Relationship Modeling", *ACM SIGSoft Software Engineering Notes* 16 (2), April 1991, pp. 31-32.

[146] King, K.A., *Findings of the CASE Review.* ISG Research Report 58, Whitbread plc, London, May 1991.

[147] King, S.F., "Making CASE Work". Unpublished Research Proposal, Advanced Technology Centre, University of Warwick, Coventry, April 1992.

[148] King, S.F., "The Quality Gap: A Case Study in Information System Development Quality and Productivity Using CASE Tools", in Spurr, K. and Layzell, P. (eds.), *CASE: Current Practice, Future Prospects.* John Wiley & Sons Ltd, Chichester, 1992, pp. 35-54.

[149] King, S.F. and Warren, G.J., *A Study of the Use of Systems Engineering Tools in a System Development Project.* University of Warwick & Rover Advanced Technology Centre Report, Coventry, July 1991.

[150] Kitchenham, B.A., "An Evaluation of Software Structure Metrics", *Proceedings COMPSAC '88*, 1988, pp. 369-376.

[151] Kitchenham, B.A., "Empirical Studies of the Assumptions Underlying Software Cost Estimation Models", *To appear, Information and Software Technology*, 1992.

[152] Kitchenham, B.A. and Pickard, L.M., "Towards a Constructive Quality Model Part II: Statistical techniques for modelling software in the ESPRIT REQUEST project", *Software Engineering Journal*, July 1987, pp. 114-126.

[153] Kitchenham, B.A., Pickard, L.M. and Linkman, S.J., "An Evaluation of Some Design Metrics", *Software Engineering Journal*, January 1990, pp. 50-58.

[154] Kitchenham, B.A. and Walker, J.G., "A Quantitative Approach to Monitoring Software Development", *Software Engineering Journal*, January 1989, pp. 2-13.

[155] Knafl, G.J. and Sacks, J., "Software Development Effort Prediction Based on Function Points", *Proceedings COMPSAC '86*, Chicago IL, 1986, pp. 319-324.

[156] Lanphar, R., "Quantitative Process Management in Software Engineering, A Reconciliation Between Process and Product Views", *Journal of Systems and Software* 12, 1990, pp. 243-248.

[157] Lee, P.-T. and Tan, K.P., "Modelling of Visualised Data-flow Diagrams Using Petri Net Model", *Software Engineering Journal*, January 1992, pp. 4-12.

[158] Lennselius, B., "Software Complexity and its Impact on Different Software Handling Processes", *Proceedings IEE 6th Conference on Software Engineering for*

*Telecommunication Switching Systems*, Eindhoven, 1986.

[159] Lew, K.S., Dillon, T.S. and Forward, K.E., "Software Complexity and Its Impact on Software Reliability", *IEEE Transactions on Software Engineering* 14 (11), November 1988, pp. 1645-1655.

[160] Li, H.F. and Cheung, W.K., "An Empirical Study of Software Metrics", *IEEE Transactions on Software Engineering* 13 (6), June 1987, pp. 697-708.

[161] Lin, C.-Y., "Systems Development With Application Generators: An End User Perspective", *Journal of Systems Management*, April 1990, pp. 32-36.

[162] Lloyd's, *Code of Practice for Computer System Development Projects.* Issue 4, Systems Development Division, Lloyd's of London, August 1989.

[163] Lloyd-Williams, M. and Beynon-Davies, P., "Knowledge Based CASE Tools for Database Design", in Spurr, K. and Layzell, P. (eds.), *CASE: Current Practice, Future Prospects.* John Wiley & Sons Ltd, Chichester, 1992, pp. 205-222.

[164] Londeix, B., *Cost Estimation for Software Development.* Addison-Wesley, Wokingham, 1987.

[165] Longworth, H.D., Ottenstein, L.M. and Smith, M.R., "The Relationship Between Program Complexity And Slice Complexity During Debugging Tasks", *Proceedings COMPSAC '86*, Chicago IL, 1986, pp. 383-389.

[166] Lor, K.-W.E. and Berry, D.M., "Automatic Synthesis of SARA Design Models from System Requirements", *IEEE Transactions on Software Engineering* 17 (12), December 1991, pp. 1229-1240.

[167] Macdonald, I., *The Finer Points of Data Modelling.* Presented to the BCS CASE Specialist Group, London, 5 September 1991.

[168] MacDonell, S.G., "An Examination of Techniques for the Measurement and Improvement of Software Development Productivity". Honours Dissertation, Department of Quantitative and Computer Studies, University of Otago, Dunedin, November 1988.

[169] Magel, K.I., "A Theory of Small Program Complexity", *ACM SIGPLan Notices* 17 (3), March 1982, pp. 37-45.

[170] Mantha, R.W., "Data Flow and Data Structure Modeling for Database Requirements Determination: A Comparative Study", *MIS Quarterly*, December 1987, pp. 531-545.

[171] March, S.T. (ed.), *Entity-Relationship Approach.* IEEE Computer Society Press, Washington, 1988.

[172] Maria, A., "CASE Technology: Today's Reality", *Journal of Systems Management*, February 1991, pp. 18, 26.

[173] Martin, J. and McClure, C., *Software Maintenance - The Problem And Its Solutions.* Prentice-Hall, Englewood Cliffs NJ, 1983.

[174] Mason, R.E.A. and Carey, T.T., "Prototyping Interactive Information Systems", *Communications of the ACM* 26 (5), May 1983, pp. 347-354.

[175] McCabe, T.J., "A Complexity Measure", *IEEE Transactions on Software Engineering* 2 (4), December 1976, pp. 308-320.

[176] McFadden, F.R. and Hoffer, J.A., *Data Base Management.* Benjamin/Cummings Publishing (2nd ed.), Menlo Park CA, 1988.

[177] Melton, A., Gustafson, D.A., Bieman, J.M. and Baker, A.L., "A Mathematical Perspective for Software Measures Research", *Software Engineering Journal*, September 1990, pp. 246-254.

[178] Miller, G.A., "The Magical Number Seven, Plus or Minus Two. Some Limits on our Capacity for Processing Information", *Psychological Review* 63, 1956, pp. 81-97.

[179] Modell, M.E., "The Entity-Relationship Approach as a Tool for Application Analysis", in Chen, P.P. (ed.), *Entity-Relationship Approach - The Use of ER Concept in Knowledge Representation.* IEEE Computer Society Press, Washington, 1985, pp. 123-130.

[180] Munson, J.C. and Khoshgoftaar, T.M., "The Dimensionality of Program Complexity", *Proceedings 11th International Conference on Software Engineering*, Pittsburgh PA, 1989, pp. 245-253.

[181] Munson, J.C. and Khoshgoftaar, T.M., "Applications of a Relative Complexity Metric for Software Project Management", *Journal of Systems and Software* 12, 1990, pp. 283-291.

[182] Myers, G.J., *Composite/Structured Design.* Van Nostrand Reinhold, New York, 1978.

[183] Myers, J.P. Jr, "The Complexity of Software Testing", *Software Engineering Journal*, January 1992, pp. 13-24.

[184] Myrvold, A., "Data Analysis for Software Metrics", *Journal of Systems and*

*Software* 12, 1990, pp. 271-275.

[185] Naib, F.A., "An Application Of Software Science To The Quantitative Measurement Of Code Quality", *ACM SIGMetrics Performance Evaluation Review* 11 (3), Fall 1982, pp. 101-128.

[186] Naulls, R., "User Experience with a CASE Tool Set (IEW and GAMMA)", *Proceedings 11th New Zealand Computer Conference*, 1989, pp. 309-318.

[187] NCC, *Software Engineering - A Case-Based Introduction For Managers.* The National Computing Centre Ltd, Manchester, 1988.

[188] Necco, C.R., Gordon, C.L. and Tsai, N.W., "Systems Analysis and Design: Current Practices", *MIS Quarterly*, December 1987, pp. 461-476.

[189] Nejmeh, B.A., "NPATH: A Measure Of Execution Path Complexity And Its Applications", *Communications of the ACM* 31 (2), February 1988, pp. 188-200.

[190] Nelson, M.S., "Computer Aided Software Engineering (CASE)". Paper 645, Master of Business Administration, University of Otago, Dunedin, February 1990.

[191] Norman, R.J. and Chen, M., "Working Together to Integrate CASE (Guest Editors' Introduction)", *IEEE Software*, March 1992, pp. 13-16.

[192] Norman, R.J. and Nunamaker, J.F. Jr, "CASE Productivity Perceptions of Software Engineering Professionals", *Communications of the ACM* 32 (9), September 1989, pp. 1102-1108.

[193] Norusis, M.J., *SPSS/PC+ V3.0 Update Manual.* SPSS Inc., Chicago IL, 1988.

[194] Oman, P.W. and Cook, C.R., "Design and Code Traceability Using a PDL Metrics Tool", *Journal of Systems and Software* 12, 1990, pp. 189-198.

[195] Ottenstein, L.M., "Predicting Numbers Of Errors Using Software Science", *ACM SIGMetrics Performance Evaluation Review* 10 (1), 1981, pp. 157-167.

[196] Paulson, D. and Wand, Y., "An Automated Approach to Information Systems Decomposition", *IEEE Transactions on Software Engineering* 18 (3), March 1992, pp. 174-189.

[197] Porter, A.A. and Selby, R.W., "Empirically Guided Software Development Using Metric-Based Classification Trees", *IEEE Software*, March 1990, pp. 46-54.

[198] Prather, R.E., "An Axiomatic Theory of Software Complexity Measure", *The Computer Journal* 27 (4), 1984, pp. 340-347.

[199] Protsko, L.B., Sorenson, P.G., Tremblay, J.P. and Schaefer, D.A., "Towards the Automatic Generation of Software Diagrams", *IEEE Transactions on Software Engineering* 17 (1), January 1991, pp. 10-21.

[200] Ramamoorthy, C.V., Prakash, A., Tsai, W.-T. and Usuda, Y., "Software Engineering: Problems and Perspectives", *Computer*, October 1984, pp. 191-209.

[201] Ramamoorthy, C.V., Tsai W.-T., Yamaura, T. and Bhide, A., "Metrics Guided Methodology", *Proceedings COMPSAC '85*, Chicago IL, 1985, pp. 111-120.

[202] Ratcliff, B. and Rollo, A.L., "Adapting Function Point Analysis to Jackson System Development", *Software Engineering Journal*, January 1990, pp. 79-84.

[203] Rinaldi, D., "Case: Front-End Tools – Getting Beyond Drawings", *Software Magazine* 8 (5), April 1988, pp. 51-58.

[204] Robinson, K., "Putting the SE into CASE", in Spurr, K. and Layzell, P. (eds.), *CASE: Current Practice, Future Prospects.* John Wiley & Sons Ltd, Chichester, 1992, pp. 1-20.

[205] Rodriguez, V. and Tsai, W.-T., "Software Metrics Interpretation Through Experimentation", *Proceedings COMPSAC '86*, Chicago IL, 1986, pp. 368-374.

[206] Rodriguez, V. and Tsai, W.-T., "Evaluation Of Software Metrics Using Discriminant Analysis", *Proceedings COMPSAC '87*, 1987, pp. 245-251.

[207] Rodriguez, V. and Tsai, W.-T., "A Tool for Discriminant Analysis and Classification of Software Metrics", *Information and Software Technology* 29 (3), April 1987, pp. 137-151.

[208] Roland, J., "Software Metrics", *Computer Language (USA)*, June 1986, pp. 27-33.

[209] Roman, G.-C., "A Taxonomy of Current Issues in Requirements Engineering", *Computer*, April 1985, pp. 14-21.

[210] Rosenquist, C.J., "Entity Life Cycle Models and their Applicability to Information Systems Development Life Cycles", *The Computer Journal* 25 (3), 1982, pp. 307-315.

[211] Rousseeuw, P.J. and Leroy, A.M., *Robust Regression and Outlier Detection.* John Wiley & Sons, New York, 1987.

[212] Rudolph, E.E., *Productivity In Computer Application Development.* Working

Group Report, University of Auckland, Auckland, 1983.

[213] Rudolph, E.E., *Measuring Information Systems.* Seminar Guide and Additional Notes, Auckland, 1987.

[214] Rummens, N. and Sucher, R., "CASE: Competitive Edge", *Systems International* 17 (3), March 1989, pp. 31-34.

[215] Sallis, P.J., "Functionality And Performance With 4GL System Development", *Proceedings VAX Forum '86*, Wellington, 1986.

[216] Sallis, P.J., "Quality Assurance And Productivity Enhancement", *Proceedings Rutherford Conference*, New Plymouth, 1988.

[217] Samson, W.B., Nevill, D.G. and Dugard, P.I., "Predictive Software Metrics Based on a Formal Specification", *Information and Software Technology* 29 (5), June 1987, pp. 242-248.

[218] Senn, J.A., *Analysis and Design of Information Systems.* McGraw-Hill, New York, 1985.

[219] Senn, J.A., *Analysis and Design of Information Systems.* McGraw-Hill (2nd ed.), Singapore, 1989.

[220] Shen, V.Y., Conte, S.D. and Dunsmore, H.E., "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support", *IEEE Transactions on Software Engineering* 9 (2), March 1983, pp. 155-165.

[221] Shepperd, M., "A Critique of Cyclomatic Complexity as a Software Metric", *Software Engineering Journal*, March 1988, pp. 30-36.

[222] Shepperd, M., "An Evaluation of Software Product Metrics", *Information and Software Technology* 30 (3), April 1988, pp. 177-188.

[223] Shepperd, M., "Design Metrics: An Empirical Analysis", *Software Engineering Journal*, January 1990, pp. 3-10.

[224] Shepperd, M. and Ince, D.C., "Metrics, Outlier Analysis and the Software Design Process", *Information and Software Technology* 31 (2), March 1989, pp. 91-98.

[225] Shepperd, M. and Ince, D.C., "Design Metrics and Software Maintainability: An Experimental Investigation", *Software Maintenance* 3, 1991, pp. 215-232.

[226] Shoval, P. and Even-Chaime, M., "Data Base Schema Design: An Experimental Comparison Between Normalization And Information Analysis", *ACM SIGBDP*

*Data Base*, Spring 1987, pp. 30-39.

[227] Snyders, J., "The CASE of the Artful Dodgers", *Infosystems*, March 1988, pp. 28-32.

[228] Sorensen, P.F., "In Search Of Program Complexity", *ACM SIGPLan Notices* 23 (2), February 1988, pp. 28-35.

[229] Spaccapietra, S. (ed.), *Entity-Relationship Approach*. IEEE Computer Society Press, Washington, 1987.

[230] Spratt, L. and McQuilken, B., "Applying Control-Flow Metrics To COBOL", *Proceedings 1987 Conference on Software Maintenance*, Austin TX, 1987, pp. 38-44.

[231] Stamps, D., "CASE vs. 4GLs", *Datamation*, 15 August 1989, pp. 29-32.

[232] Statland, N., "Payoffs Down the Pike: A CASE Study", *Datamation*, April 1989, pp. 32-33, 52.

[233] Stevens, O.B., *Project Sizing Methodology*. Development Assurance Services Report, Lloyd's Bank, London, August 1990.

[234] Sumner, M.R., "How Should Applications Be Developed? An Analysis of Traditional, User, and Microcomputer Development Approaches", *ACM SIGBDP Data Base*, Fall 1985, pp. 25-33.

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

[236] Symons, C.R., *Software Sizing and Estimating: Mk II FPA (Function Point Analysis)*. John Wiley & Sons Ltd, Chichester, 1991.

[237] Takahashi, M. and Kamayachi, Y., "An Empirical Study of a Model for Program Error Prediction", *IEEE Transactions on Software Engineering* 15 (1), January 1989, pp. 82-86.

[238] Tan, K.P., Chua, T.S. and Lee, P.-T., "AUTO-DFD: An Intelligent Data Flow Processor", *The Computer Journal* 32 (3), 1989, pp. 194-201.

[239] Tao, Y. and Kung, C., "Formal Definition and Verification of Data Flow Diagrams", *Journal of Systems and Software* 16, 1991, pp. 29-36.

[240] Tate, G., "Management, CASE and the Software Process", *Proceedings 12th New Zealand Computer Conference*, Dunedin, 1991, pp. 247-256.

[241] Tate, G. and Docker, T.W.G., "A Rapid Prototyping System Based On Data Flow Principles", *ACM SIGSoft Software Engineering Notes* 10 (2), April 1985, pp. 28-34.

[242] Tate, G. and Verner, J., "Software Metrics for CASE Development", *Proceedings COMPSAC '91*, Tokyo, 1991, pp. 565-570.

[243] Tate, G. and Verner, J., "Approaches to Measuring Size of Application Products with CASE Tools", *Information and Software Technology* 33 (9), November 1991, pp. 622-628.

[244] Tate, G., Verner, J. and Jeffery, R., "CASE: A Testbed for Modeling, Measurement and Management", *Communications of the ACM* 35 (4), April 1992, pp. 65-72.

[245] Teorey, T.J., Yang, D. and Fry, J.P., "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relational Model", *ACM Computing Surveys* 18 (2), June 1986, pp. 197-222.

[246] Troy, D.A. and Zweben, S.H., "Measuring the Quality of Structured Designs", *Journal of Systems and Software* 2, 1981, pp. 113-120.

[247] Tsai, J.J.-P. and Ridge, J.C., "Intelligent Support for Specifications Transformation", *IEEE Software*, November 1988, pp. 28-35.

[248] Tsai, W.-T., Lopez, M.A., Rodriguez, V. and Volovik, D., "An Approach To Measuring Data Structure Complexity", *Proceedings COMPSAC '86*, Chicago IL, 1986, pp. 240-246.

[249] Tse, T.H. and Pong, L., "Towards a Formal Foundation for DeMarco Data Flow Diagrams", *The Computer Journal* 32 (1), 1989, pp. 1-11.

[250] Turnbull, C., "Translating Creativity into Working Systems", *Computing Canada* 14 (7), 31 March 1988, pp. 24-25, 36.

[251] Vargo, J. and Kong, Y.S., "CASE Productivity in New Zealand", *New Zealand Journal of Computing* 2 (1), December 1990, pp. 55-67.

[252] Verner, J. and Tate, G., "A Model for Software Sizing", *Journal of Systems and Software* 7, 1987, pp. 173-177.

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

[254] Verner, J., Tate, G., Jackson, B. and Hayward, R.G., "Technology Dependence in Function Point Analysis: A Case Study and Critical Review", *Proceedings 11th*

*International Conference on Software Engineering*, Pittsburgh PA, 1989, pp. 375-382.

[255] Vessey, I., Jarvenpaa, S.L. and Tractinsky, N., "Evaluation of Vendor Products: CASE Tools as Methodology Companions", *Communications of the ACM* 35 (4), April 1992, pp. 90-105.

[256] Vessey, I. and Weber, R., "Some Factors Affecting Program Repair Maintenance: An Empirical Study", *Communications of the ACM* 26 (2), February 1983, pp. 128-134.

[257] Waguespack, L.J. Jr and Badlani, S., "Software Complexity Assessment: An Introduction and Annotated Bibliography", *ACM SIGSoft Software Engineering Notes* 12 (4), October 1987, pp. 52-71.

[258] Webster, D.E., "Mapping the Design Information Representation Terrain", *Computer*, December 1988, pp. 8-23.

[259] Weissman, L., "Psychological Complexity Of Computer Programs: An Experimental Methodology", *ACM SIGPLan Notices* 9 (6), June 1974, pp. 25-36.

[260] Weyuker, E.J., "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering* 14 (9), September 1988, pp. 1357-1365.

[261] Williamson, M., "Learning to Practice What They Preach", *CIO* 5, April 1988, pp. 22-29.

[262] Wilson, M.L., "The Measurement Of Usability", in Chen, P.P. (ed.), *Entity-Relationship Approach To Systems Analysis And Design*. North-Holland, Amsterdam, 1980, pp. 75-101.

[263] Woodfield, S.N., Shen, V.Y. and Dunsmore, H.E., "A Study of Several Metrics for Programming Effort", *Journal of Systems and Software* 2, 1981, pp. 97-103.

[264] Worsley, L.M., "Project Management and the Use of Metrics in a Fourth Generation Environment", *Proceedings 8th European Oracle Users Group Conference*, Madrid, April 1990.

[265] Wrigley, C.D. and Dexter, A.S., "A Model for Measuring Information System Size", *MIS Quarterly*, June 1991, pp. 245-257.

[266] Yadav, S.B., "Control and Definition Modularization: An Improved Software Design Technique for Organizing Programs", *IEEE Transactions on Software Engineering* 16 (1), January 1990, pp. 92-99.

[267] Yau, S.S. and Collofello, J.S., "Some Stability Measures for Software Maintenance", *IEEE Transactions on Software Engineering* 6 (6), November 1980, pp. 545-552.

[268] Zahniser, R.A., "The Perils of Top-Down Design", *ACM SIGSoft Software Engineering Notes* 13 (2), April 1988, pp. 22-24.

# Appendices

## A.1 Development Site Response Results

The general applicability of the results of the empirical segment of this study is somewhat restricted by the small data sets available for analysis. This situation is a direct result of the lack of response and commitment from the software development sites and automated tool vendors contacted in the first year of this research. The degree of response obtained is shown in the following two spreadsheet printouts.

| Development Sites: | UK CASE Sites | NZ CASE Sites | 4GL Sites | Devmt Sites | TOTALS |
|---|---|---|---|---|---|
| Number of sites | 156 | 80 | 21 | 83 | 340 |
| Number of repeats | 69 | 0 | 0 | 3 | 72 |
| Total letters | 225 | 80 | 21 | 86 | 412 |
| Number of replies | 45 | 33 | 9 | 39 | 126 |
| Number of calls | 9 | 0 | 0 | 17 | 26 |
| Total responses | 54 | 33 | 9 | 56 | 152 |
| Percent responses | 24.00% | 41.25% | 42.86% | 65.12% | 36.89% |
| Further contact | 21 | 6 | 0 | 12 | 39 |
| Preliminary agreement | 13 | 3 | 0 | 6 | 22 |
| Final agreement | 7 | 0 | 0 | 3 | 10 |
| Minimal use of tool(s) | 5 | 10 | 1 | 4 | 20 |
| Data unavailable | 11 | 3 | 4 | 5 | 23 |
| Just starting with tools | 10 | 3 | 0 | 7 | 20 |
| Plan off-shelf/third party | 0 | 5 | 0 | 0 | 5 |
| Have rejected tools | 0 | 2 | 0 | 0 | 2 |
| Mgmt/Takeover | 0 | 1 | 0 | 1 | 2 |
| Training only | 1 | 1 | 0 | 0 | 2 |
| No resources | 4 | 2 | 1 | 2 | 9 |
| Security/Confidentiality | 6 | 1 | 1 | 2 | 10 |
| Still evaluating tools | 3 | 0 | 0 | 2 | 5 |
| No CASE/4GL use | 10 | 0 | 1 | 11 | 22 |
| Number sent on | 4 | 2 | 1 | 2 | 9 |
| No reason | 6 | 3 | 3 | 3 | 15 |

| Vendors and Others: | CASE/4GL Vendors Distrib. | Consult. | Indep. Orgns | User Group Reps. | TOTALS |
|---|---|---|---|---|---|
| Number of sites | 44 | 13 | 6 | 4 | 67 |
| Number of repeats | 6 | 0 | 0 | 0 | 6 |
| Total letters | 50 | 13 | 6 | 4 | 73 |
| Number of replies | 13 | 4 | 4 | 1 | 22 |
| Number of calls | 4 | 1 | 0 | 0 | 5 |
| Total responses | 17 | 5 | 4 | 1 | 27 |
| Percent responses | 34.00% | 38.46% | 66.67% | 25.00% | 36.99% |
| Number sent on | 5 | 0 | 1 | 0 | 6 |
| List provided | 3 | 1 | 2 | 0 | 6 |

From the first set of figures it can be seen that a total of 340 distinct development sites were contacted, in an effort to obtain a large representative sample. The overall rate of response, however, was just 37%. Particularly disappointing was the low response rate from U.K. CASE product users, at just 24% of those contacted. This was in spite of the fact that the research was 'free', it was to take up very little of the participating organisations' time and resources, and was hopefully to lead to outcomes beneficial to those organisations. Furthermore, of the 152 replies that were received, continued response was only maintained by thirty-nine, leading to a group of twenty-two granting preliminary agreement and to a final sample of just ten sites.

A number of reasons were given by responding sites as to their unwillingness or inability to take part in the study. A total of forty sites stated that their automated tool usage was minimal or had only just begun. Moreover, twenty-two others cited no tool use; surprisingly, ten of these twenty-two were from CASE product user lists. Whether this absence of tool use was because these sites had abandoned the tools, or because the organisations were mistakenly on the lists, however, is unclear. King [147] states that CASE is still relatively new, with widespread use only being achieved in the last two to three years. This may help to explain the poor response level achieved and the low usage in those organisations that did respond. Another contributor, although not explicitly cited in the replies, may have been a lack of success with the products. Although CASE is becoming increasingly accepted (Chen and Norman [43]), success is unfortunately not inherent with the purchase of automated products. In fact, failures are relatively common in situations where organisations have purchased a product as the solution to their development problems, but have failed to address the equally important issues of effective training, management commitment and the adoption of appropriate work methods. Organisations, however, that have directed their attention to these problems as well as

to their technical requirements have achieved success with automated tools. It was a requirement of this study that the participating organisations were committed and relatively mature product users—this may have inadvertently precluded a large number of those contacted. Experience from the United States, however, where successful CASE usage would appear to be increasing (Burkhard and Jenster [36]; Glass [97]), would suggest that this may not be the situation for long.

Despite the opportunity for 'free' research nine respondents cited lack of resources as a reason for their non-participation. Rather more frustrating was the issue of security/confidentiality—ten sites chose not to take part because they felt they were unable to release details of their systems for analysis. Although in many situations this apprehension would be justified, it was thought that the current study would not have caused too many security fears, given that details of the operational systems were not required and only the specifications were used as the basis for assessment. Furthermore it was stressed to the organisations that any written agreements required by the sites would be complied with.

Finally an issue of major concern was the unavailability of the requested project management data in twenty-three of the responding sites, including almost half of the replying U.K. CASE sites. It would appear that routine data collection relating to project development progression is still not a high priority in many organisations, in spite of increasing emphasis being placed on project durations and budget control. Some respondents remarked that they needed proof that collection was actually worthwhile before they were prepared to allocate resources to it. However, this produces a cyclic problem: no data means that proof cannot be provided; a lack of proof discourages sites to collect the data. It can only be hoped that the prediction of Tate and Verner [243], that project management data will also be automatically collected in a CASE environment, will prove to be correct. Perhaps in this way consistent data will become available for ongoing analysis.

The second set of figures above, relating to approaches to product vendors and other organisations, reveal an equally disappointing response rate at just 37% of those contacted. Of the sixty-seven distinct organisations originally contacted only six provided a list of product users. Although this project was clearly an academic one, the objectives were of a practical nature and it was hoped at the outset that the outcomes would be of real use to the software development community. Inclusion of this observation in the contact letters, however, failed to produce the anticipated degree of response.

The relative success of this project was only made possible by the involvement of the ten sites that did take part—the results obtained in the analysis will hopefully be of use to them. More importantly, however, it is hoped that the results may encourage other sites to participate in future studies of this type, because only with real-world assistance can we hope to provide real-world solutions.

## A.2 Examples of Data and Statistical Analysis Output Listings

EXAMPLE OF RAW DATA
**********

Sample One: Macroanalysis-Effort
==========

Transaction Measures
==========

| TCR | TRE | TUP | TDE |
|---|---|---|---|
| 3 | 29 | 2 | 3 |
| 17 | 62 | 5 | 4 |
| 11 | 17 | 20 | 0 |
| 20 | 47 | 10 | 5 |
| 18 | 54 | 12 | 10 |
| 5 | 13 | 5 | 5 |
| 14 | 67 | 22 | 2 |
| 14 | 28 | 11 | 12 |
| 26 | 158 | 20 | 9 |
| 36 | 145 | 81 | 11 |
| 7 | 250 | 5 | 0 |
| 103 | 241 | 103 | 103 |
| 16 | 107 | 20 | 6 |

Data Model Measures
==========

| TESDM | TDEPD | TEP | TDECD | TEC | TAU | TAC |
|---|---|---|---|---|---|---|
| 4 | 4 | 29 | 4 | 8 | 19 | 60 |
| 14 | 14 | 62 | 11 | 26 | 61 | 152 |
| 10 | 6 | 17 | 9 | 31 | 203 | 61 |
| 21 | 13 | 47 | 14 | 35 | 110 | 206 |
| 18 | 15 | 29 | 18 | 40 | 152 | 188 |
| 6 | 4 | 13 | 4 | 15 | 180 | 149 |
| 9 | 9 | 54 | 9 | 38 | 144 | 207 |
| 16 | 15 | 28 | 13 | 37 | 236 | 220 |
| 23 | 22 | 114 | 22 | 55 | 255 | 758 |
| 40 | 35 | 145 | 33 | 128 | 577 | 821 |
| 32 | 29 | 250 | 8 | 12 | 137 | 1766 |
| 77 | 77 | 241 | 77 | 309 | 1287 | 1080 |
| 32 | 32 | 107 | 16 | 42 | 572 | 1271 |

System Effort Measures
==========

| DESIGN | PROGRAM | AN/DES | PROG/UT | TOTAL |
|---|---|---|---|---|
| 6.0 | 2.5 | 6.0 | 3.5 | 11.5 |
| 9.5 | 4.0 | 9.5 | 6.5 | 21.0 |
| 12.0 | 4.5 | 12.0 | 7.0 | 26.0 |
| 15.5 | 4.0 | 15.5 | 5.5 | 27.5 |
| 11.0 | 19.5 | 20.0 | 19.5 | 39.5 |
| 39.5 | 30.0 | 51.5 | 30.0 | 81.5 |
| 40.0 | 26.5 | 40.0 | 73.5 | 113.5 |
| 56.0 | 12.5 | 56.0 | 46.5 | 119.5 |
| 13.5 | 136.0 | 38.5 | 136.0 | 189.5 |
| 88.5 | 41.0 | 88.5 | 67.0 | 216.5 |
| 50.0 | 60.0 | 50.0 | 140.0 | 290.0 |
| 70.0 | 40.0 | 220.0 | 80.0 | 315.0 |
| 119.5 | 185.0 | 165.5 | 185.0 | 355.5 |

EXAMPLE OF STATISTICAL ANALYSIS OUTPUT LISTINGS
**********

Sample One: Macroanalysis-Effort
==========

Correlation
==========

N of cases: 13 1-tailed Signif: * - .01 ** - .001

Data Model Measures:

| Correlations: | DESIGN | PROGRAM | AN_DES | PROG_UT | TOTAL |
|---|---|---|---|---|---|
| TESDM | .5614 | .2859 | .8521** | .4304 | .7466* |
| TDEPD | .5736 | .3184 | .8804** | .4630 | .7685* |
| TEP | .4780 | .3658 | .6314 | .6278 | .8147** |
| TDECD | .3789 | .1190 | .7839** | .1992 | .5310 |
| TEC | .3942 | .0543 | .7882** | .1524 | .5078 |
| TAU | .6346* | .3048 | .9372** | .3739 | .7000* |
| TAC | .6390* | .6428* | .6152 | .8471** | .9160** |
| TOOLS | -.0408 | .3629 | .0849 | .5450 | .4474 |
| TOMLS | .6668* | .3170 | .8935** | .4333 | .7617* |
| TMMLS | .7596* | .7365* | .4957 | .6370* | .5838 |

RANK ALL /PRINT NO.

| Correlations: | RDESIGN | RPROGRAM | RAN_DES | RPROG_UT | RTOTAL |
|---|---|---|---|---|---|
| RTESDM | .6602* | .6529* | .6437* | .6300 | .7868** |
| RTDEPD | .6804* | .6869* | .6970* | .7025* | .8264** |
| RTEP | .4512 | .5923 | .4017 | .6327 | .6547* |
| RTDECD | .4132 | .4138 | .4738 | .3581 | .5152 |
| RTEC | .5495 | .5447 | .6154 | .5110 | .6209 |
| RTAU | .7198* | .6850* | .8077** | .6319 | .7473* |
| RTAC | .8077** | .8033** | .7582* | .8736** | .9341** |
| RTOOLS | .1509 | .4250 | .2829 | .5093 | .4715 |
| RTOMLS | .6648* | .5722 | .6374* | .5495 | .7418* |
| RTMMLS | .5237 | .3254 | .3881 | .3344 | .4291 |

Intercorrelation
==========

| Correlations: | TESDM | TDEPD | TAU | TAC | TMLS |
|---|---|---|---|---|---|
| TESDM | 1.0000** | .9921** | .9223** | .6695* | .9893** |
| TDEPD | .9921** | 1.0000** | .9346** | .6713* | .9843** |
| TAU | .9223** | .9346** | 1.0000** | .4896 | .9084** |
| TAC | .6695* | .6713* | .4896 | 1.0000** | .7258* |
| TMLS | .9893** | .9843** | .9084** | .7258* | 1.0000** |
| TAM | .8792** | .8860** | .7857** | .9241** | .9131** |
| TIDM | .9749** | .9706** | .9321** | .7071* | .9750** |
| TSDM | .9916** | .9858** | .9337** | .6955* | .9871** |

Normality Tests
==========

TRE

| | Statistic | df | Significance |
|---|---|---|---|
| Shapiro-Wilks | .8528 | 13 | .0363 |
| K-S (Lilliefors) | .2440 | 13 | .0332 |

TOTAL

| | Statistic | df | Significance |
|---|---|---|---|
| Shapiro-Wilks | .8833 | 13 | .0876 |
| K-S (Lilliefors) | .1787 | 13 | > .2000 |

```
Classification
==============

TDSCR and TOTAL (Normal):
-------------------------
Measure TDSCR:
                      -4.550          48.000          105.550
Systems:                 | 1 2 3 4 5 6 7 | 10 11 13       | 12
                         | 8 9           |                |
                         |               |                |

Systems:    1            | 2 3 4 5 6 7 8 | 9 10           | 11 12 13
                         |               |                |
                         |               |                |
                      16.888          138.962         261.036
Measure TOTAL:

Direct mapping for systems: 2 3 4 5 6 7 8 10 12
Number of systems: 13      Correct classification: 9/13 = 69.2%

Direct mapping for outliers: 12
Number of response outliers: 3     Correct identification: 1/3
Number of incorrectly identified outliers: 0

TDSCR and TOTAL (Boxplot):
--------------------------
Measure TDSCR:
                     32.000          48.500          85.000
Systems:   1 2 3 4 5 6 | 7 8 9         | 10 11          | 12 13
                       |               |                |
                       |               |                |

Systems:   1 2 3 4 5 6 | 7 8 9 10      | 11             | 12 13
                       |               |                |
                       |               |                |
                    113.500         226.500         315.000
Measure TOTAL:

Direct mapping for systems: 1 2 3 4 5 6 7 8 9 11 12 13
Number of systems: 13      Correct classification: 12/13 = 92.3%

Direct mapping for outliers: 12 13
Number of response outliers: 2     Correct identification: 2/2
Number of incorrectly identified outliers: 0
```

Estimation
==========

Estimation - An_Des Effort Using TDSCR (Constant Term)

LEAST SQUARES REGRESSION
************************

| VARIABLE | COEFFICIENT | STAND. ERROR | T - VALUE | P - VALUE |
|---|---|---|---|---|
| TDSCR | 1.11132 | .15724 | 7.06746 | .00002 |
| CONSTANT | 6.11806 | 10.95432 | .55851 | .58769 |

SUM OF SQUARES = 9013.06300
DEGREES OF FREEDOM = 11
SCALE ESTIMATE = 28.62463
COEFFICIENT OF DETERMINATION (R SQUARED) = .81952
THE F-VALUE = 49.949 (WITH 1 AND 11 DF) P - VALUE = .00002

LEAST MEDIAN OF SQUARES REGRESSION
**********************************

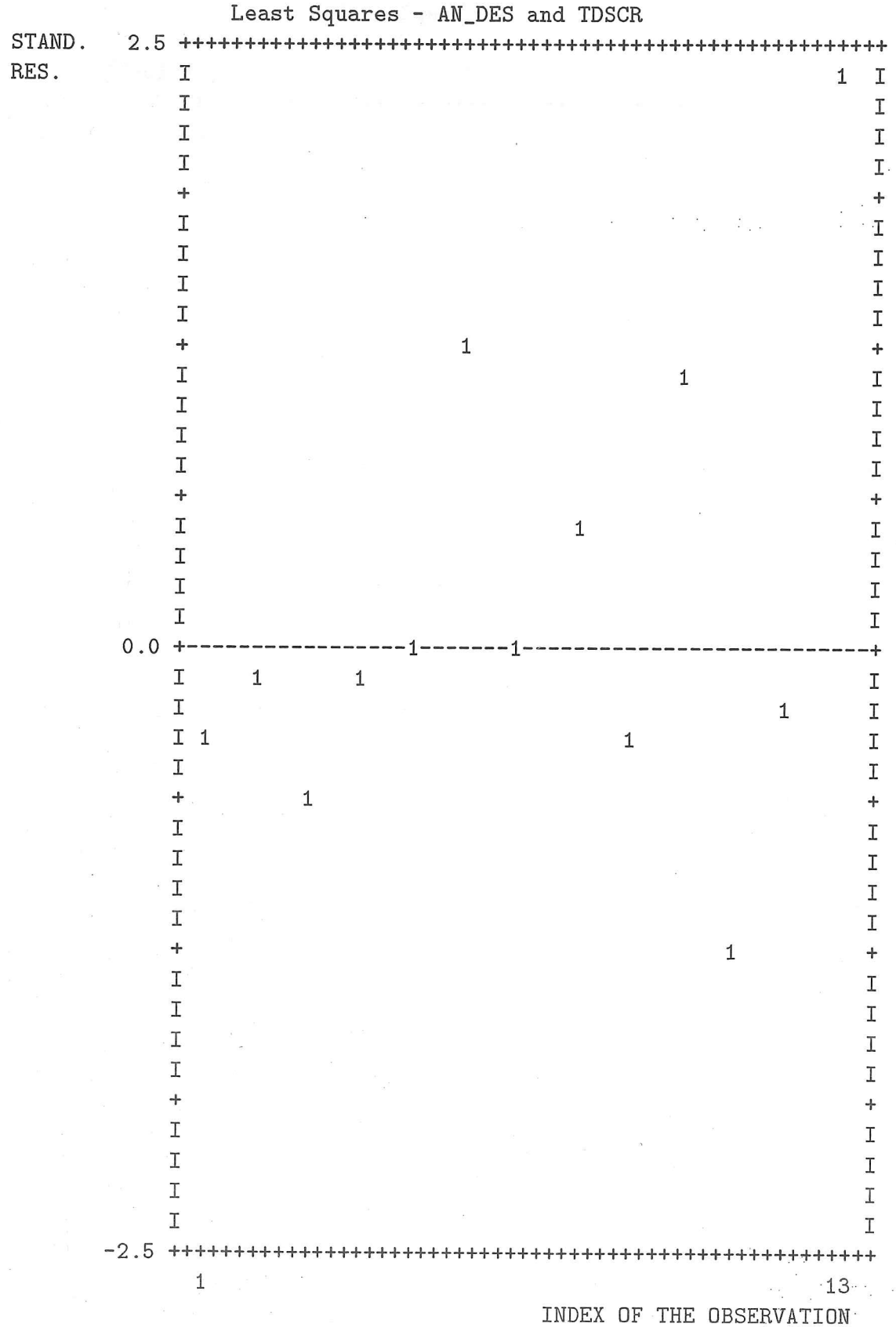
| VARIABLE | COEFFICIENT |
|---|---|
| TDSCR | 1.92857 |
| CONSTANT | -14.96428 |

FINAL SCALE ESTIMATE = 16.16273
COEFFICIENT OF DETERMINATION = .80201

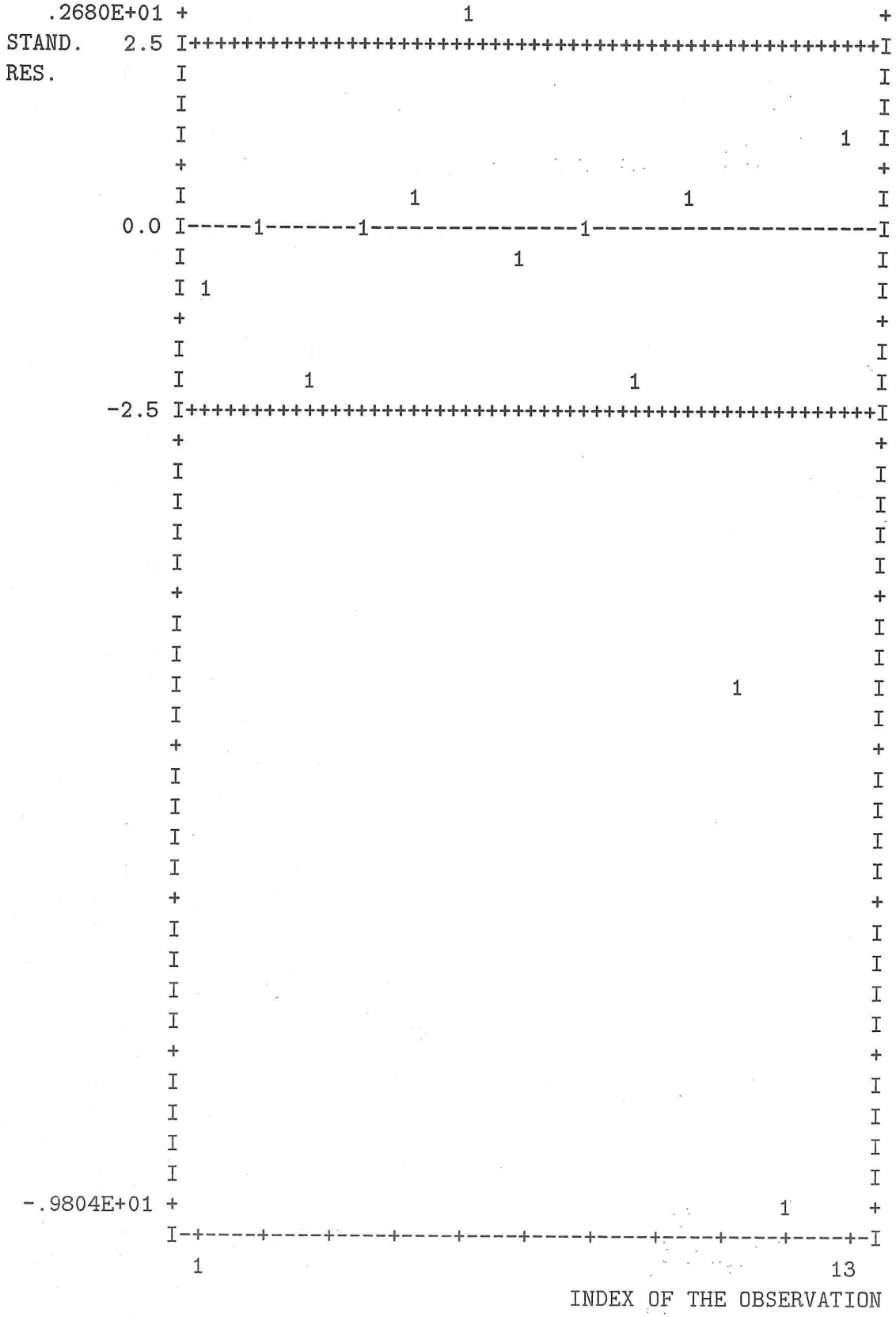
REWEIGHTED LEAST SQUARES BASED ON THE LMS
*****************************************

| VARIABLE | COEFFICIENT | STAND. ERROR | T - VALUE | P - VALUE |
|---|---|---|---|---|
| TDSCR | 2.09009 | .22082 | 9.46496 | .00001 |
| CONSTANT | -24.44990 | 8.75114 | -2.79391 | .02342 |

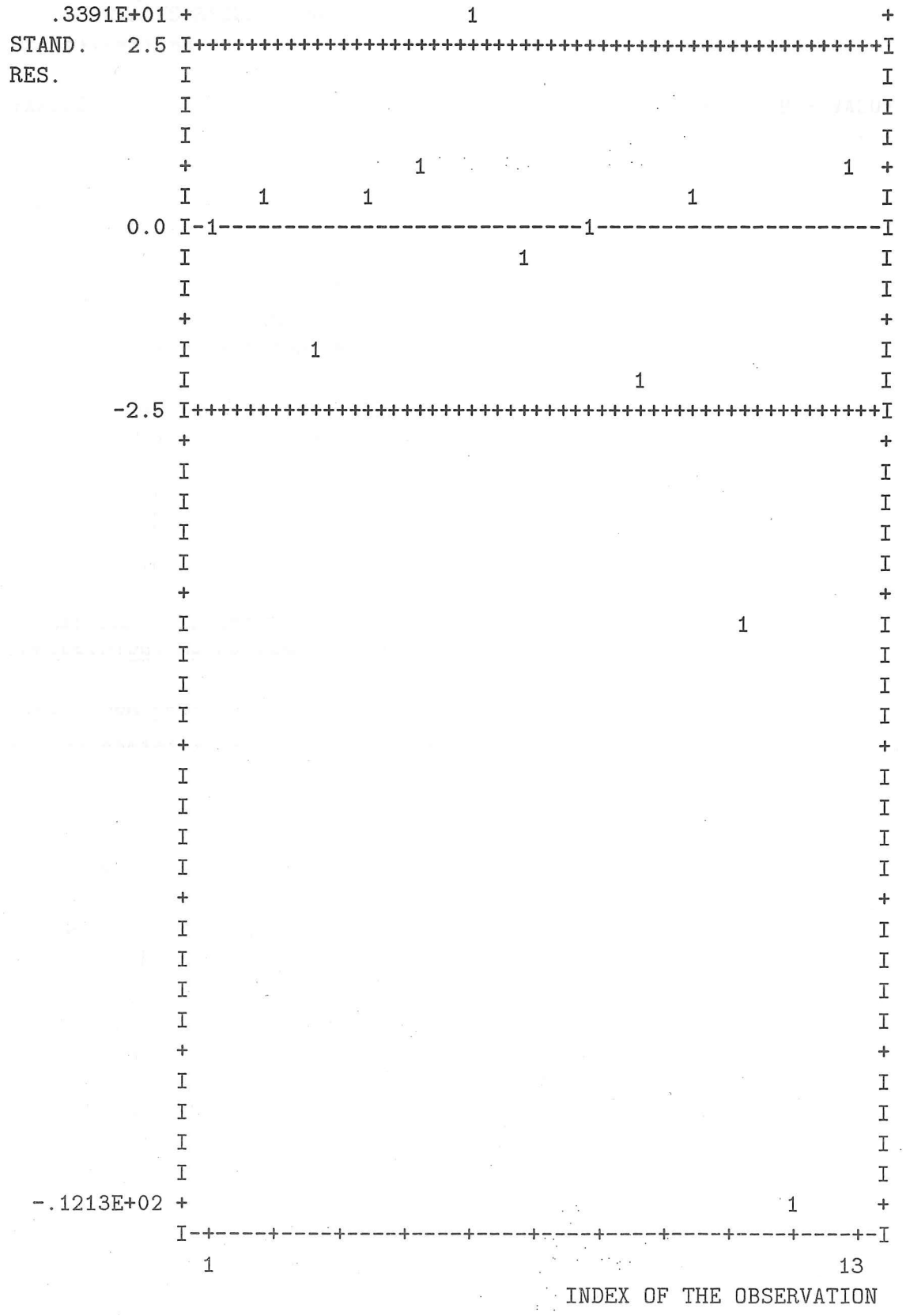
WEIGHTED SUM OF SQUARES = 1800.76200
DEGREES OF FREEDOM = 8
SCALE ESTIMATE = 15.00318
COEFFICIENT OF DETERMINATION (R SQUARED) = .91802
THE F-VALUE = 89.585 (WITH 1 AND 8 DF) P - VALUE = .00001
THERE ARE 10 POINTS WITH NON-ZERO WEIGHT.
AVERAGE WEIGHT = .76923


Least Squares - AN_DES and TDSCR
STAND. RES.
2.5
0.0
-2.5
1
13
INDEX OF THE OBSERVATION

Least Median Squares - AN_DES and TDSCR



Reweighted Least Squares - AN_DES and TDSCR



Estimation - An_Des Effort Using TDSCR (Through Origin)

LEAST SQUARES REGRESSION
**********

| VARIABLE | COEFFICIENT | STAND. ERROR | T - VALUE | P - VALUE |
|---|---|---|---|---|
| TDSCR | 1.17183 | .11065 | 10.59080 | .00000 |

SUM OF SQUARES = 9268.64700
DEGREES OF FREEDOM = 12
SCALE ESTIMATE = 27.79186
COEFFICIENT OF DETERMINATION (R SQUARED) = .90335
THE F-VALUE = 112.165 (WITH 1 AND 12 DF) P - VALUE = .00000

LEAST MEDIAN OF SQUARES REGRESSION
**********

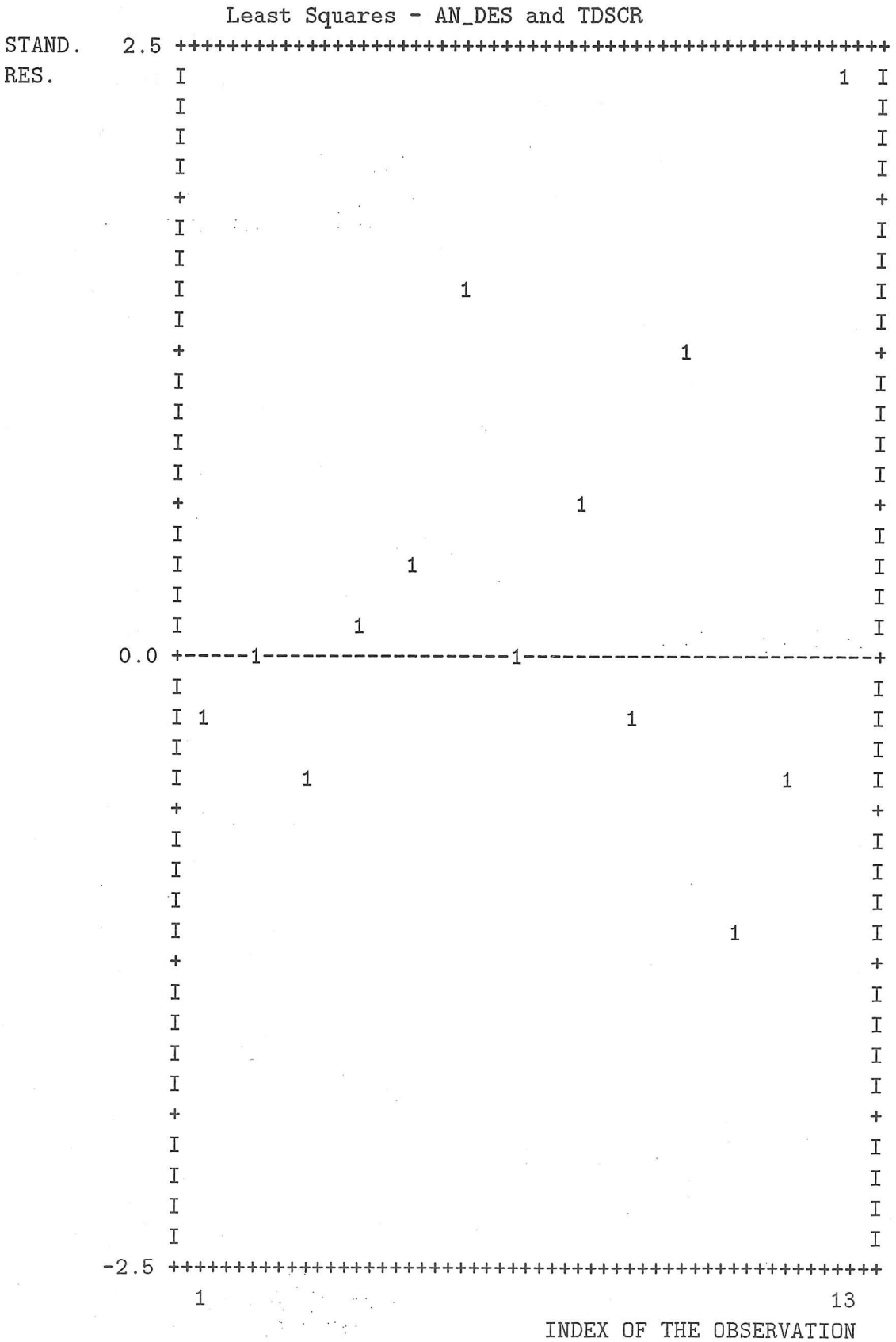
| VARIABLE | COEFFICIENT |
|---|---|
| TDSCR | .79167 |

FINAL SCALE ESTIMATE = 23.06456
COEFFICIENT OF DETERMINATION = .84874

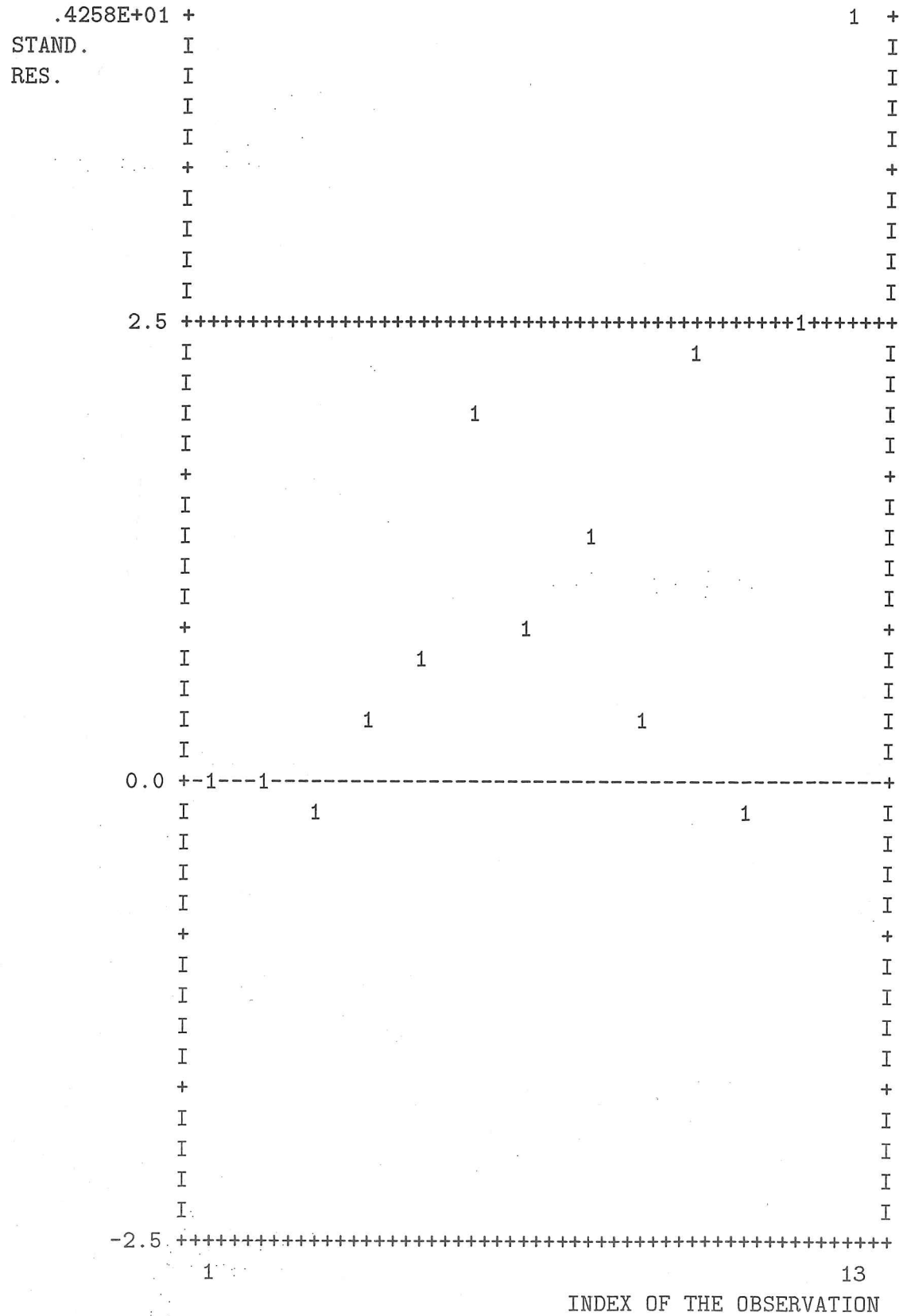
REWEIGHTED LEAST SQUARES BASED ON THE LMS
**********

| VARIABLE | COEFFICIENT | STAND. ERROR | T - VALUE | P - VALUE |
|---|---|---|---|---|
| TDSCR | 1.05155 | .15964 | 6.58716 | .00006 |

WEIGHTED SUM OF SQUARES = 4357.38900
DEGREES OF FREEDOM = 10
SCALE ESTIMATE = 20.87436
COEFFICIENT OF DETERMINATION (R SQUARED) = .78336
THE F-VALUE = 36.159 (WITH 1 AND 10 DF) P - VALUE = .00013
THERE ARE 11 POINTS WITH NON-ZERO WEIGHT.
AVERAGE WEIGHT = .84615


Least Squares - AN_DES and TDSCR
STAND. RES.
2.5
0.0
-2.5
1
13
INDEX OF THE OBSERVATION

Least Median Squares - AN_DES and TDSCR



Reweighted Least Squares - AN_DES and TDSCR