AN EXPLORATION OF PARAMETRIC SOFTWARE COST ESTIMATING MODELS FOR JACKSON SYSTEMS DEVELOPMENT

ANTHONY LEES ROLLO

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

January 1995

The University of Aston in Birmingham

An Exploration of Parametric Software Cost Estimating Models for Jackson Systems Development

Anthony Lees Rollo

Doctor of Philosophy

1995

Summary

Most parametric software cost estimation models used today evolved in the late 70's and early 80's. At that time, the dominant software development techniques being used were the early 'structured methods'. Since then, several new systems development paradigms and methods have emerged, one being Jackson Systems Development (JSD). As current cost estimating methods do not take account of these developments, their non-universality means they cannot provide adequate estimates of effort and hence cost. In order to address these shortcomings two new estimation methods have been developed for JSD projects one of these methods JSD-FPA, is a top-down estimating method, based on the existing MKII function point method. The other method, JSD-COCOMO, is a sizing technique which sizes a project, in terms of lines of code, from the process structure diagrams and thus provides an input to the traditional COCOMO method.

The JSD-FPA method allows JSD projects in both the real-time and scientific application areas to be costed, as well as the commercial information systems applications to which FPA is usually applied. The method is based upon a three-dimensional view of a system specification as opposed to the largely data-oriented view traditionally used by FPA. The method uses counts of various attributes of a JSD specification to develop a metric which provides an indication of the size of the system to be developed. This size metric is then transformed into an estimate of effort by calculating past project productivity and utilising this figure to predict the effort and hence cost of a future project. The effort estimates produced were validated by comparing them against the effort figures for six actual projects. The JSD-COCOMO method uses counts of the levels in a process structure chart as the input to an empirically derived model which transforms them into an estimate of delivered source code instructions.

Key Words: Cost estimation, sizing metrics, JSD, FPA, COCOMO.

Acknowledgments

I wish to take this opportunity to thank my supervisor, Mr B Ratcliff, for his endurance, guidance, support and encouragement during the period of this research. I would also like to thank Mr G Rule, who has provided encouragement, assistance and useful discussions on different occasions. Finally, I would wish to express my gratitude to Michael Jackson Systems Ltd, Abbey National and SEMA Scientific who assisted with the provision of data and encouragement.

I would also like to express my gratitude to my employers during the period, the University of Luton and Suffolk College, who have provided both the time and finances to allow me to pursue this research.

List of Contents

List of Tables

List of Figures

Chapter 1

Introduction

"It is very difficult to make predictions,

especially about the future"

Chinese Proverb

Introduction

Accurate and reliable estimates of the resource requirements for a software project are difficult to obtain, even if the task of arriving at them is approached in a systematic manner, "no accurate method exists"(Ferrentino, 1981). The difficulty of estimating is compounded by the number of factors impinging upon the final outcome of the project involved (Mohanty, 1981). The problem is made more intractable because of the difficulty in quantifying the individual contributions of these factors to the overall cost of the project. Software project managers have at their disposal several techniques that have been developed since the early 1970's, some of which have improved the reliability of the estimating process. The techniques fall into several basic categories (Macro & Buxton, 1987), though some 'techniques' are more correctly regarded as not being estimates at all (DeMarco, 1982). The subject of this research falls into the category of methods which use parametric models of software cost/effort to produce estimates.

1.1 The Software Engineering Context

The term *software engineering* was coined in the late 1960's and can be traced back to a 1968 NATO conference held in Garmisch in what was then West Germany (Naur *et al*, 1976). The conference was held to discuss what was termed the 'software crisis' The difficulties being experienced by the software industry which gave rise to this crisis were founded in the development of the new 'third generation' computers These machines were very much more powerful than their predecessors and they allowed the development of ever more demanding software applications The computing power of these new machines resulted in a demand for applications that had previously been unrealisable (Sommerville, 1992)

In attempting to satisfy this demand the software industry soon found that the techniques which had sufficed until that time were now hopelessly inadequate. The task of constructing the much larger application software being demanded was not amenable to a simple scaling up of the current methods (Ratcliff, 1987). The situation was rather as if the development of new materials meant that those boat builders who had previously constructed small sailing craft out of wood were now being required to construct super tankers

A number of major projects were delivered late — sometimes by years. The resulting software also exhibited other deficiencies such as being

- well over budget
- unsatisfactory in meeting users' requirements
- impossible to maintain
- difficult to use.

Thus software development was experiencing a crisis: hardware costs were falling rapidly while software costs were rising at an ever growing rate. These trends have remained since that time, and world-wide the current cost of software development and maintenance is huge. Up-to-date figures are difficult to obtain, however Boehm (1987) claims that in 1985 software costs were in excess of $140 billion and growing at around 12% per year. From a sample of 600 organisations in the United States 35% admitted to having at least one runaway problem (Boehm, **ibid.**).

The 1968 NATO conference, and its 1969 follow up in Rome, were held to discuss the software crisis and to promote the development and spread of techniques to deal with the problems in the construction of ever larger software systems. The trend to larger systems has continued. A project to develop a command and control system for Royal Navy submarines has been estimated at around 1 million lines of Ada (France & Lawton, 1987). The resulting development of new methods and techniques has created the discipline of Software Engineering.

The tribulations facing this relatively young discipline do not abate — the 'software problem' is still with us in a very real way (Ami, 1992). Examples of the continuing difficulties are legion (Business, 1991). The demand for software of ever increasing size and complexity continues to outstrip the abilities of the software industry to satisfy them. There remains a world-wide shortage of skilled software developers (Conte *et al*, 1986; Boehm, 1983). The discipline of software engineering does not yet fully pervade the software development

industry. "Many individuals and companies still develop software haphazardly" (Pressman, 1992). As a result much of today's software is still delivered late and is of poor quality (Ami, 1992). Even in those areas of industry where software engineering techniques are practised, problems remain. A now classic example from the area of defence avionics is the RAF Nimrod Airborne Early Warning system. AEW Nimrod was eventually abandoned by the British Government but not before its costs had reached almost £1 billion (Times, 1988), several times its original estimate, the major cause being put down to delays with the software for the on-board RADAR system. At the time of its cancellation the Nimrod was three years late on an original seven year delivery schedule, and the projection was that at least a further four years would be required before delivery to the original Air Staff requirements could be achieved.

Software Engineering

Several authors have provided definitions of software engineering and an exploration of these definitions will be useful. Boehm (1984) defines software engineering as follows:

> *"Software Engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation."*

This definition of software engineering includes rather more than simply programming. The phrase 'useful to man' implies issues such as a social awareness of the results of software development and also a need to deliver software in a timely and cost effective manner.

Ratcliff (1987) defines software engineering in the following manner:

> *"Software Engineering is the disciplined utilisation of a systematic, coherent set of principles, tools and techniques within a planned organisational framework, the objective of which is to achieve the on-schedule, cost-effective development of quality software for a diverse range of non-trivial applications and environments."*

In this definition the need for timeliness and cost effectiveness are made overt along with such factors as quality and planning.

Fenton (1991) regards Software Engineering as being:

> *"...the term used to describe the collection of techniques concerned with applying an engineering approach to the construction of software products. By 'engineering' approach we mean managing, costing, planning, modelling, analysing, designing, implementing, testing and maintaining."*

Again, in Fenton's definition there is an overt reference to issues of costing, planning and managing the development of software artefacts.

What emerges from an examination of these various definitions, which are by no means exhaustive, is that software engineering is not concerned solely with the activities of analysis, design, programming, testing and maintenance. The management aspects of developing a software project are seen to be as important as, and wholly integrated with, the computer science aspects of the engineering of software artefacts.

The requirement in various definitions of software engineering for timeliness, cost effectiveness and management of the process highlights the most intractable areas of software development. Among the major difficulties which remain are those of providing accurate estimates of the cost and staffing required to construct large software products. Many large software projects are

still running out of control: costs are often double the original estimate (Londeix, 1987), and the ability of project managers to measure and forecast the progress of their projects remains a problem (Conte *et al.*, 1986). This usually results in these projects being not only late and over-budget but frequently producing software of poor quality.

1.2 The Need for Estimating

Before considering why we need to develop estimates of software costs it will be useful to define what is meant in this context by an 'estimate'. The *Concise Oxford Dictionary* defines an estimate as follows:

- an approximate judgement
- a contractor's statement of the sum for which he will undertake specified work.

This latter definition provides us with one reason for developing cost estimates: software developers contracted to produce a software artefact are able to state how much they will charge. Costing a software development project is important whether the cost is to be charged to an external client or is for internal use within an organisation. A software cost estimate also acts as a link between the software developer and the financial analyst. The financial analyst will use standard accounting techniques such as cost-benefit analysis, break-even analysis or make-or-buy analysis. There is little probability of making sound financial analysis of a software project unless it is preceded by a reliable estimate of the cost of developing the software (Boehm, 1981).

Software developers are well aware that when large software projects are started they may steadily get out of hand (Brooks, 1974, 1975,1982; DeMarco, 1982; Londeix, 1987). Without good estimates the software project manager has no sound basis for determining how much time and effort[1] each software phase and activity should consume (Boehm, 1981; Londeix, 1987). Consequently the project manager has no way of being sure that the project is proceeding to plan — in other words it is out of control from the start. With no reliable way of generating estimates of cost, analysts are unable to perform well founded hardware/software trade-offs so that often a cheap hardware solution may be rejected for a more expensive software solution (Boehm, 1981). Finally, without recognised sound methods for estimation, software project personnel have no rational basis for defending their requirements for budget and schedule to senior management who may propose budgets and schedules which are unrealistic.

The Oxford dictionary definition given earlier regards an estimate as an opinion about cost. However, the software project manager also has a need to predict both project schedule and staffing requirements, preferably for each software development phase. Therefore, most cost estimation techniques in current use provide estimates of total staff effort and project schedule broken down across the various life-cycle phases.

[1] Many authors use the terms 'cost' and 'effort' as if they are synonyms. Software development is highly labour intensive and therefore the major cost of a software project is determined by the staff work effort required.

1.3 The Metrics Context

All engineering disciplines are characterised by their basis in methods "that are underpinned by models and theories" (Fenton, 1991). The civil engineer can appeal to beam theory when constructing load bearing surfaces, the electrical engineer to the laws of Ohm and Kirchoff when designing circuits. The models and theories of traditional engineering disciplines such as electrical or civil engineering have their foundation in the classical scientific method. The basis of the scientific method is that of observation and experiment, and is critically dependant upon measurement (Dunham, 1983, Fenton, 1991). In order to develop theoretical models, the scientist and engineer need measurement so that results of observation or experimentation may be discussed in a common context and be replicated by colleagues within that discipline.

The discipline of Software Engineering is no different — if it is to be considered a real engineering discipline then it must be based upon reliable theoretical models. The theoretical models of software engineering must themselves be rooted in a rigorous foundation of observation and experiment. Indeed, measurement must be a core skill of the software engineer. Not only does the very foundation of the discipline depend upon measurement but, as in other engineering disciplines, so must the day-to-day activities of the software engineer.

The continuing crisis discussed earlier is at least partially attributable to the failure of software engineering management. This failure is due to the inability to handle properly the interaction of several factors such as the complexity of

applications, the abilities of personnel, and the characteristics of computer systems on which the software is to be developed and installed. The software engineering manager thus has a need for theoretical models of the development process that may be used to assist in handling the effects of these diverse factors. A major need within the discipline of software engineering is for the development and application of techniques and tools that can assist project management in forecasting and controlling project costs.

1.4 Metrics for Estimating

Before it is possible to estimate the resources required for a software project, it is necessary to have some quantifiable features or *metrics* on which to base any predictions of expected effort or progress. Therefore the use of software metrics is inextricably linked to the problem of providing reliable estimates. Software metrics are quantifiable features of a software system, such as lines of code, number of faults in acceptance testing, work hours, etc. Metrics may be classified as being one of two types (Kitchenham & Linkman 1989; Conte et al., 1986; Ince, 1989). These are *product metrics* and *process metrics* that are described in Conte *et al.* (1986) as follows:

- Process metrics: quantify attributes of the development process and of the development environment.
- Product metrics: are measures of the software product.

Process metrics include *resource* metrics such as the degree of programmer experience, the level of tool support in development, etc. Product metrics are attributes of the product itself such as size, total defect counts, and so on. There can be uncertainty as to which category a metric belongs. For example, the

number of faults found in testing will be affected by the type and rigour of the testing process as well as by the nature of the product. Metrics should be classified according to whichever factor is considered to have the greatest significance.

Both process and product metrics can be further classified as *result* or *predictor* metrics (Ince, 1989; DeMarco, 1982)

- Result metrics quantify some attribute of a completed project, or part project.
- Predictor metrics are used to make an estimate of the value of a result metric.

As DeMarco points out, any theorised relationships between a predictor and a result metric are purely speculative until sufficient empirical research has been carried out to support the theory. To be of greatest use as a predictor of development effort, a metric that can be derived at the earliest possible stage of the product development cycle is the most desirable.

1.5 The JSD Context

Jackson System Development (JSD) (Jackson, 1983; Cameron, 1986 and 1989; Sutcliffe, 1988) is a software development method which belongs to the operational paradigm. Apart from JSD, other documented examples of the operational approach include PAISley (Zave, 1982, 1986), GIST (Balzer *et al.*, 1982; Feather, 1982) and Me-Too. However, in the United Kingdom, JSD is now used in a number of organisations as a system development method in the areas of both embedded real-time and commercial data processing applications.

An operational method constructs a specification that is based on a model of the application domain and can produce the behaviour of the system it specifies either by direct execution or as the result of transformation (Bass, 1992). While an operational specification could indeed be directly executed, should a suitable virtual machine exist, it would be extremely inefficient and resource hungry; because of this, attempts at direct execution have been limited to the development of prototypes. In common with other operational methods of system development, JSD does not conform to the conventional paradigm of software development characterised by the waterfall life-cycle model (Zave, 1984). Both the conventional and operational approaches start with the identification of a problem or need which is thought to be solvable by a computer based-system. The approaches differ when the system developers begin to formulate a solution.

The conventional approach starts with a requirements definition phase, followed by a system specification phase (also variously referred to as: outline design, product design), the outcome of which is a system specification that regards the system as a black box and describes all the external aspects of its behaviour. This is followed by the system design phase during which the system is described, typically as a series of black box modules whose external behaviour will satisfy the system specification. Finally, in the implementation phase, the modules are described in terms of their internal structures such that they conform to the external behaviour specified for that module in the system design. These structures are then coded into some computer language and each module is integrated into the final system. It should be noted that in the design

stage the modules which are specified are intended to operate in the final implementation environment so that their specification takes account of the run time configuration and constraints.

In the operational approach, the system developers formulate a system specification which describes the internal structures necessary to produce the behaviour required of the system. The structures are described in some implementation-independent form, but the structures specified will generate the behaviour of the desired final system. Not only are the specification structures independent of implementation considerations but they also reflect the problem domain "they are chosen for modifiability and human comprehension without regard to any implementation characteristic whatsoever" (Zave, 1984). The next phase in the operational approach is to transform the problem-oriented specification into an implementation specific description. The transformations applied are correctness preserving in that they maintain the external behaviour of the system, though the internal structures are altered or supplemented in such a way that the description now relates to the specific implementation environment required. For this reason the same operational specification could be transformed to suit a variety of implementation environments. Finally the transformed specification is implemented in some computer language. The module structure of the final implementation may be very different to that of the original specification.

The JSD variant of the operational approach consists of three main stages which are characterised as:

- *Modelling* In this first stage objects of relevance to the system are identified in the real world along with the important events associated with them. In JSD, entities and their associated events, or 'actions', are structured into what can be termed 'entity life histories'.

- *Network* During this stage the functional requirements of the system are added to the basic model of the previous stage. The result of this stage is a network of asynchronous concurrent processes, with accompanying text based descriptions.

- *Implementation* Finally, the abstract implementation-independent specification arrived at in the two preceding stages is transformed into a practical system implementation. It is during this stage that the constraints imposed by the characteristics of the target environment are addressed (language, virtual machine, scheduling requirements, etc.) to realise a system of acceptable performance.

The description of the operational specification that has been given could make it seem no more than the design of conventional methods, but this is not so. The operational specification is completely independent of considerations of implementation, no reference is made to timing requirements or resource constraints as is generally the case in a conventional design.

Operational methods such as JSD have significantly different life-cycles to that described by the waterfall model and its variants (Bass, 1992; Rollo & Ratcliff, 1988, 1990; Zave, 1984). From the point of view of the estimator, the two

approaches have very different work load patterns. In the operational approach more effort is required to produce the specification than would be true of the specification in a conventional method. Hence the operational approach has a greater front loading in development effort than do the conventional techniques. Conventional techniques start with a concern for the architecture of the system to be built, usually by modelling the structure of the new system directly from the existing system. In contrast operational techniques initially concentrate on modelling the application domain, thus constructing a system which is based on a real-world view of the application. It is this difference in approach that makes an operational method interesting from the point of view of the estimator and raises the question, given that conventional estimating techniques were derived empirically from conventional development projects, can they be adapted to predict development effort for an operational development method such as JSD?

1.6 The Need for New Estimation Methods

As stated earlier, the subject of this thesis is that category of methods which use parametric models of software cost/effort to produce estimates. Parametric methods, as the name implies, use attributes of the system to be developed as inputs to a mathematical model of the predictive relationship between system size and the effort (hence cost) required to produce the system. Strictly, these methods do not estimate the total cost of system development. For example, they make no claim to predict the cost of hardware, or the environment required to develop the system, they merely attempt to predict the staff effort required.

Several parametric methods exist to assist project management in estimating the cost of building a software artefact. Those methods in current use were evolved in the late 70s and early 80s when the dominant software development

techniques being employed in industry were those developed in the mid 70s, such as Structured Design (Yourdon, 1975) Structured Analysis and Design Technique (Ross, 1977). These methods relied heavily on the traditional waterfall model of system development activity (Royce, 1970). Since that time, several new system development paradigms have emerged that do not conform to the waterfall model, among them being the object oriented paradigm (Booch, 1983) as well as the operational paradigm discussed earlier; methods belonging to these paradigms began to be widely used in the mid 80s. Therefore, as current estimating methods do not take account of the different process models that have emerged to accompany these new approaches, they cannot provide estimates of effort broken down across the system development stages. In respect of a JSD project though, they may be able to estimate the overall development effort required to complete the project. It is this non-applicability of the current estimating methods that requires the development of further models suitable for a development method such as JSD.

A further strong reason for exploring estimation allied to a specific development technique is that methods which have claimed general applicability may be less accurate when applied to the specific technique when compared to an estimation model designed around the technique's distinguishing features. Certainly, all current estimation models exhort the user to calibrate the model to their own particular environment to allow for variations in the effect of the various cost drivers and other modifiers (Boehm, 1981; Kemerer, 1987). However, it may be that a significant reduction in the need for re-calibration can be achieved by developing a model tied to a particular development technique.

1.7 Thesis Structure

Chapter 2 reviews current estimating methods, and tentatively identifies those which appear to offer scope to be adapted to suit the development method JSD. The following chapter gives a detailed exploration of operational methods and JSD in particular, and considers adaptations of candidate estimation methods for JSD. Chapter 4 discusses top-down estimating methods for JSD and some of the disadvantages of the most suitable current method (FPA), suggesting modifications to allow FPA to take advantage of specific JSD features. The fifth chapter explores the development of a bottom-up method of estimation in a JSD context, paying particular attention to the need to develop a method that does not rely upon an initial estimate of the final delivered source lines of code. Chapter 6 is an evaluation of the actual method developed through research, including a review of the experience of its partial use in industry. Finally, Chapter 7 presents conclusions and outlines further work that might usefully be carried out, both in developing automatic tool support for the method and extending it to allow the incorporation of risk management techniques.

Chapter 2

Estimating Methods

Introduction

This chapter reviews estimating methods. The chapter starts with an overview of estimating practices and a discussion of the general nature of parametric methods, then overviews specific parametric methods before proceeding to an examination of several candidate line-of-code based methods. The chapter then examines two commonly used variants of a function-based method called 'function points', before finally contrasting the candidate line-of-code based method COCOMO with the function-based methods.

2.1 Overview of Estimating Practices

The software industry uses many techniques for estimating the cost of software development, though some of these in current use should be more correctly characterised as guesses or non-estimates (DeMarco, 1982). Macro & Buxton (1987) provide a taxonomy of estimating practices. These practices are given below along with a brief description:

- *Analogy involving expert judgement on the basis of known representative samples*

  This technique consists of experienced software developers selecting previously developed whole or part systems that are analogous to the

system being estimated. The estimate is then based upon the experience with the chosen analogies. The principle difficulty with this method is that there is a high risk of false analogy in that either the scope or complexity of the previous system is not truly analogous to the one being estimated.

The method of analogy has been explored as part of the MERMAID project (Cowderoy, 1990, Jurek, 1992). Members of the project team have devised an algorithmic machine-based method for capturing the relevant parts of the analogy and performing a prediction of the effort required for the system under investigation.

- *Analogy based on more or less representative experience of non-specialists*

  This method relies upon a user, usually a manager or salesperson, basing an estimate on a simplistic analogy. An example would be if a salesperson said "we did a stock control system for ABC plc and that took a year to complete so that's how long your stock control system will take". Clearly the probability of such an estimate being accurate is low, as no account will have been taken of differences in the requirements of the two systems. The principle difference between this technique and the first one is that in the first the analogy is drawn by technical experts who will adjust the estimate by taking account of differences in requirements between the system being estimated and the one from which the analogy is drawn.

- *Application of Parkinson's law*

Parkinson's law states that "work expands to fill the available volume". This is not an estimating method at all but is at best an attempt at justification for short time scales. Boehm (1981) gives an example of a Parkinsonian estimate as:

> *"This flight control software must fit on a 65K word machine therefore its size will be roughly 65,000 words. It must be done in 18 months, and there are 10 people available to work on it, so the job will take roughly 180 man months."*

Boehm contends that the system above finally grew to 127,000 words and required a total of 32 months and 550 man months. Macro & Buxton consider its continued use as an indication of the parlous state of software development

- *Bidding 'price-to-win'*

This is the practice of deciding what the market will bear or what a competitor is likely to bid and then bidding below this price to win a contract. When bidding on the basis of what the market will bear it is possible that the bid price will exceed the cost to build. On most occasions, however, the price-to-win will be far short of the cost to build. Bidding price-to-win is a perfectly legitimate business practise when attempting to break into a new market or when attempting to exclude competing businesses from breaking into your market area. The technique should always be accompanied, or at least followed, by a sound estimate so that the bidder has some idea of the cost of any underbidding.

- *Top-down requirements analysis and rudimentary definition of a 'system'*

This technique is described by Macro & Buxton (1987) as an attempt to construct a notional model of the system-to-be from the generic features and concepts extracted from a user's statement of requirements. The model is then used by 'experts' to judge the required effort to construct. Unfortunately this often proves to be extremely inaccurate as little real account can be taken of the true scope and complexity of the system to be constructed. The method is also liable to instability as changes occur within the initial user requirements. Top-down methods, however, have considerable appeal as they are applicable at early stages of the life-cycle. Perhaps the real lesson to be learnt from this brief account is that a single estimate should never be used throughout a project. As requirements are refined, so also should the estimate and indeed, as the project progresses estimating should also continue.

- *Detailed software design by software engineers, and the derivation of estimates from the 'bottom-up' using activities planning techniques based on actual tasks and staff assignment*

In bottom-up estimation the known modularity of the design and the known assignment of staff can be used to draw up detailed estimates of each activity and these accreted to provide an estimate for the entire system. Although the method can be quite accurate, the main problem is that to arrive at a point in the life-cycle where the technique may be accurately attempted, considerable effort (in the order of 35-45%) has

already been expended. Several authors (Symons, 1991; CCTA, 1991) advocate the use of a bottom-up approach alongside a top-down approach because at late life-cycle stages the bottom-up approach will give greater accuracy.

- *Parametric estimating models*

  Parametric methods are the subject of this research and will be described later.

- *Notional activities planning by software engineers based on the software taxonomy of the outline system design*

  This method is a bottom-up method that is used earlier in the life-cycle than was possible for the method based on a detailed design described earlier. The method relies upon a complete hierarchical list of the system components accompanied by an activities plan. At this point in the life-cycle the bottom level of the list will represent features and functions that require a few hundred source lines at most to realise. It is therefore possible to employ a simple project planning technique such as Project Evaluation and Review Technique (PERT) or the Critical Path Method (CPM) to derive simultaneously both the effort and time scales required. Macro & Buxton claim the method has accuracy of around 10%-20% and that the technique can be applied after only about 15% of project effort has been expended.

- *Task based estimation*

  In addition to the methods listed by Macro & Buxton, a method in common use is a bottom up method known as task based estimation. Task based estimation consists of a detailed list of all the tasks that need to be carried out in each stage of development, an estimate of the time required for each task is then drawn up based on a judgement as to the number of design or implementation deliverables to be completed during the stage (MJUK, 1989b)

2.2 Parametric Methods

The methods to be examined as a part of this research are confined to that class of methods known as *parametric* or *algorithmic* methods. A main objective of research into parametric methods is to establish some relationship between characteristics of the software to be constructed and the effort and time-scale required to realise that construction The methods rely upon certain input parameters, typically these have been measures or calculations representing the size or scope of the system, complexity, type of system, requirements, etc. which are then transformed by an empirically derived algorithm into an estimate of the effort and time-scale required to develop the system.

Most recently developed parametric methods have in general three main stages, not always applied in the same order or to the same attributes. The three stages are the *system size estimation* stage, an estimate of *modifiers* of either *size or effort*, which are based on other system measures such as schedule compression, complexity, etc., and a calculation of an *effort estimate* (see Figure 2.1). The

modifiers are applied differently in the various models. Thus a method may estimate size, calculate effort and then modify the calculation on the basis of several features that are considered to affect the effort required to construct the software. Alternative methods estimate the size, then modify this estimate on the basis of factors considered to affect the size of the task. The effort is then calculated on the basis of the adjusted size estimate. It should be noted that the modifiers of size or effort are often based on the same or similar factors.

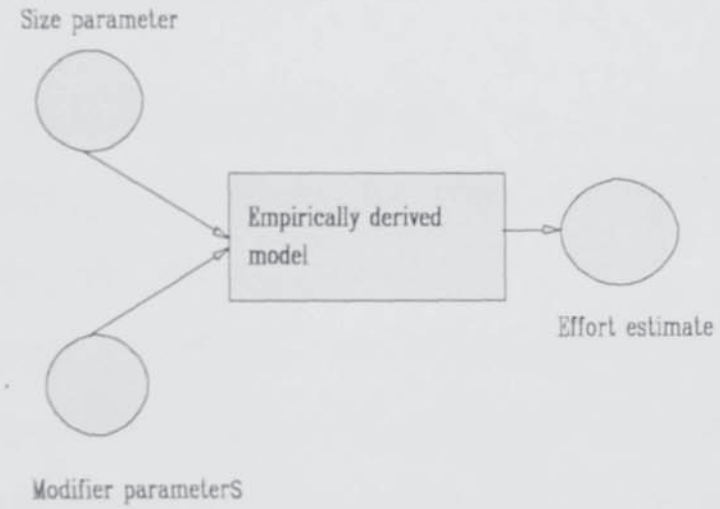


Fig. 2.1. A generic model of parametric estimating methods

Classifying Methods

Cost estimation methods are usually classified according to their initial size parameter e.g., a method may be referred to as a Line-Of-Code (LOC) based method. Parametric methods may also be classified as *top-down* or *bottom-up* which are defined as

- *Top-down* — The characteristics of the project and the environment are assessed; then, with reference to past history, a single estimate is

made for the whole project. This estimate is revised as the project and environmental characteristics are changed or refined.

- *Bottom-up* — The project is broken down into low-level activities. The effort for each activity is estimated and the total project effort obtained by summing all the individual estimates.

Initial Size Estimation

The majority of current methods use one of two basic metrics to provide an estimate of the system size:

- *An estimate of the final delivered Source Lines of Code (SLOC)*

  Example: the COnstructive COst MOdel (COCOMO) developed by Boehm (1981, 1984).

  Comment SLOC-based methods require the estimator to arrive at an estimate of the number of delivered lines of code. There are several variations in the description of which lines are to be estimated ranging from 'non-comment source statements' to 'executable source instructions' In COCOMO, for example, the definition of SLOC amounts to the non-comment source instructions, which worked well with the card-based languages such as the versions of FORTRAN and COBOL which were current until the mid eighties (one instruction usually being one card). However, there is little general agreement as to how such lines should be counted when dealing with the free format languages available today.

- *An estimate of the functionality that the system will provide to the user*

  Example: Function Point Analysis (FPA) developed by Albrecht & Gaffney (1983); see also Dreger (1989) and Symons (1988, 1990).

  Comment A few variations of function-based methods exist, and essentially these all start with a measure of system size which is based upon an estimate of the 'amount of function' to be provided by the system. The estimated functionality of the system is usually calculated by summing weighted totals of such attributes as the input and output data types, the number of files accessed, and so on.

Modifiers

Having calculated a system size metric in the function-based method the estimator proceeds to apply modifying factors to the size. In a SLOC-based method it is usual to calculate the estimate of effort and then apply the modifiers to that estimate. Modifiers of either effort or system size are derived by considering the influence on the system to be developed of various factors, such as the type of system (e.g. embedded, batch), the complexity of processing (e.g. distributed processing, performance) and personnel attributes (e.g. experience, capability). The various methods share little agreement over what constitutes the correct set of modifying factors, or indeed on the nature of their influence. However, most techniques use between 10 and 20 factors (COCOMO uses 15, FPA uses 14), though Mohanty (1981) lists some 49 factors used by the methods he surveyed that may influence the final project costs.

Effort Calculation

The final estimate is produced by using a model based upon empirical data related to past projects which has been used to derive a relationship between the size estimate, the size or effort modifiers and the effort expended. This empirical model is used to provide an estimate of the effort required for new projects.

2.3 Line-of-Code Methods

The starting point for the review of such estimating methods was the evaluation of a variety of such models contained in *Software Engineering Economics* (Boehm, 1981), in which eight models are examined that rely upon estimates of SLOC for their input; one non-SLOC method — the GRC model which will not be considered here — is also described briefly. The eight models are listed below along with Boehm's own model COCOMO:

- The Bailey-Basili Meta-Model (Bailey & Basili, 1981)
- The Boeing model (Black *et al*, 1977)
- The COCOMO model proposed by Boehm (1981,1984)
- The Doty model (Herd *et al*, 1977)
- The IBM-FSD model (Walston & Felix, 1977)
- The RCA PRICE S model (Freiman & Park, 1979)
- The Putnam SLIM model (Putnam, 1978; Putnam & Fitzsimmons, 1979)
- The System Development Corporation model (Nelson, 1966)
- The TRW Wolverton model (Wolverton, 1974)

Table 2.1. Results of the survey of SLOC based methods								
	Estimating Model							
Author	Bailey Basili	COCOMO	Doty	IBM FSD	Jensen	PRICE	SLIM	TRW - Wolverton
AbdelHamid & Madnick (1986)		x						
Baskette (1987)		x				x		
Callisen & Colborne (1984)		x						
Cuelenare *et al* (1987)		x			x		x	
Ferens (1984)		x					x	
Jensen (1981)			x		x	x		
Kemerer (1987)		x				x	x	
Kulkarni et al (1988)		x			x	x		
Londeix (1987)		x	x			x	x	x
Martin (1988)		x						
Maroune & Mili (1989)		x						
Masters (1985)		x				x	x	
Miyazaki & Mori (1985)		x						
Mohanty (1980)			x	x		x	x	x
Myers (1989)			x					
Pinsky (1984)		x					x	
Rubin (1985)		x			x		x	
Thayer (1987)	x			x				
Warburton (1983)							x	

The list is too great to permit a detailed study of each method and its ability to be adapted to suit the environment of JSD, so the current literature was reviewed to see if a single candidate method could be chosen. Several reviews of cost estimating models were encountered as well as articles describing the experiences of practitioners using the models in their own environments. The results of this survey are presented in Table 2.1.

Two of the models listed by Boehm, Boeing and SDC, did not recur in the literature surveyed; however one other SLOC-based model, Jensens's, emerged from the survey. From this survey it became apparent that the most frequently quoted methods were COCOMO (16 occurrences), SLIM (9 occurrences) and PRICE (6 occurrences). The next most frequently quoted method was the DOTY model with only four occurrences.

Review of Models

Mohanty (1981) gives a description and review of well-known models in use in the software industry. His comparison is based on a set of 49 attributes that he found in these models, which attempt to address all aspects of the software development task. Mohanty grouped the attributes into 7 factors. These factors and the number of attributes contributing to the factor are given below:

Factor	No of Attributes
Size	8
Data Base	1
Complexity	10
Type of Program	3
Documentation	3
Environment	15
Other Items	9

Table 2.2. shows a comparison, in terms of Mohanty's attributes, of the three most popular models from the survey presented in Table 2.1.

Table 2.2. Attribute comparison of the most frequently quoted cost models.

	Model and No of Attributes		
Factor	**COCOMO**	**Putnam/SLIM![2]**	**PRICE**
Size	2	2	1
Data Base	1	-	-
Complexity	7	3	1
Type of Program	3	-	2
Documentation	-	-	-
Environment	12	2	4
Other Items	4	-	1

Mohanty compared the various models in terms of the number and type of the above attributes that each model used. He then followed this by making an estimate of the cost of software produced by each model using a hypothetical system development task. The calculation produced a range of estimates varying from $362,500 to $2,766,667. Mohanty concluded that the large variation was environmental, that is, each of the models was developed for a cost database collected from a specific development environment. The database used by a model therefore reflects the work pattern and practices, of and is only suitable within, the environment for which it was developed. This finding by Mohanty is acknowledged in many of the cost estimation models that were developed subsequently and which now insist that the claimed degree of

[2] Figures given are for Putnam's Model as SLIM is a proprietary derivation of that model, and not available for comparison.

accuracy can be achieved only if a calibration database of local projects is used (Cuelenaere, 1987).

Rubin (1985) also carried out a comparison of cost models. His results are presented below in respect of COCOMO and SLIM in Table 2.3

Table 2.3 Estimation Results (Rubin, 1985)

Model	SLIM	COCOMO
Effort (Person Month)	200	363
Duration(Months)	17	23
Peak Staff	17	22

This shows considerable variation in the estimates, SLIM producing an estimate some 55% less than COCOMO. An estimate for PRICE was not given by Rubin as being a proprietary model access to the PRICE estimating Tool is required. Again, the variation is explained in terms of the need to calibrate each model for the environment in which systems development is to take place.

A similar comparison between cost models was carried out by Kemerer (1987) which showed a variation in the estimate of a similar order to the variations demonstrated by Mohanty (**ibid**.) and Ruben (**ibid**.). Kemerer concludes that models do not perform well outside their original environments and suggests that practitioners who are selecting a method should base their selection upon how easily the method can be calibrated to their environment.

Given the results of these various comparisons, it was decided that simply carrying out another similar comparison would almost certainly produce essentially the same results. The approach used to determine a candidate cost model suitable for adapting to a JSD environment (described briefly in Chapter 1) was therefore to conduct an examination of the characteristics of the three most popular models. Unfortunately, PRICE is a proprietary model and details of the underlying equations and assumptions are not available for examination. For pragmatic reasons therefore the PRICE model was rejected as a possible candidate.

2.4 SLIM

A review of cost estimation models carried out on behalf of the Ministry of Defence by McTavish (1984) concluded that SLIM is "a proprietary model that has a low visibility to the researcher.. (and) it is based on the method by Putnam" (McTavish, 1984). An examination of Putnam's method therefore gave the necessary insights into the underlying algorithms and assumptions of the SLIM model.

The Putnam method is described by Putnam (1978) and also by Gallacher (1984) and Londeix (1987). The paper by Putnam describes a model based upon a Rayleigh/Norden distribution (Norden, 1958, 1970) applied to the software life-cycle. The Putnam model takes account of both the development activities and the 'project' activities which include management, configuration control, etc., and the maintenance phase. The model was developed from the result of an extensive examination of some 19 projects developed for the US Army's Computer Systems Command. Putnam states that the model fitted well when tested against some 50 systems (also of the Computer Systems Command). He also states that another 150 systems from other operations were

examined by various authors (Jones, 1977; Sudding, 1977; Johnson, 1977), and that many of these also exhibited the same pattern of behaviour as described by Putnam's model.

Essentially, the Putnam model describes the behaviour of a software development project in terms of a set of software state variables linked by what he calls the 'software equation' which is given as

$$S = E K^{1/3} t_d^{4/3}$$

where

S = Non-Comment Source Statements (NCSS)

t_d = The time to peak staffing

K = Total manpower cost of the *Generic Cycle* (see below)

E = The environmental factor, which is calculated from past projects by a rearrangement of the software equation thus:

$$E = \frac{S}{K^{1/3} t_d^{4/3}}$$

Generic Cycle is the term used by Putnam to describe the general life-cycle model of a project and is generalised by the Rayleigh function (an example is given in Figure 2.2) for the expenditure of manpower over the whole project thus:

$$m(t) = Dt \exp\left(-\frac{t^2}{2t_d^2}\right)$$

where:

D is the gradient of the manpower curve at time t(0) and is considered to be an indication of the difficulty of a project,

The difficulty D of a project is considered by this model to increase if the project requires more manpower or an earlier completion time and is given by:

$$D = \frac{K}{t_d^2}$$

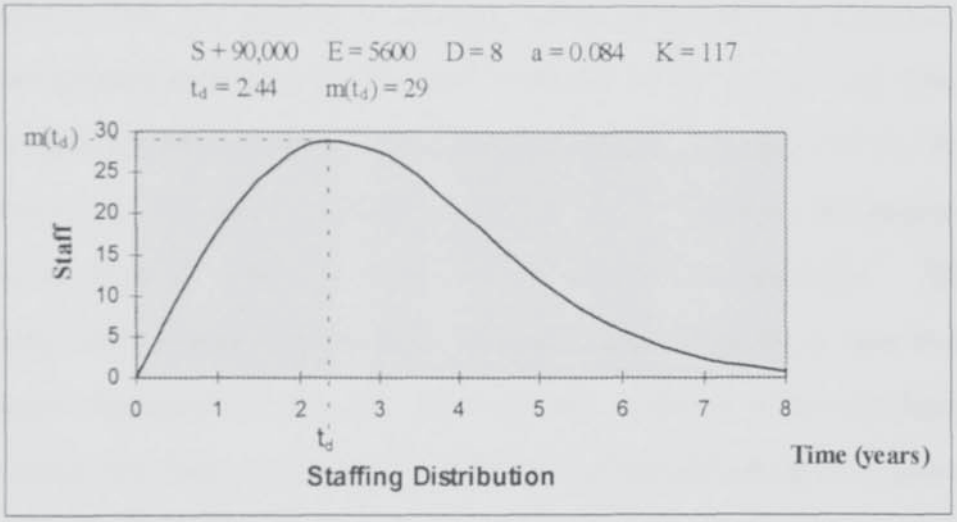


Figure 2.2 The Generic Cycle for typical Project

The claim was made (Londeix, 1987) that the Putnam model allows the dynamic behaviour of a software project to be examined, thus providing strategic choices to project management. The proponents of the method assert that it is perfectly feasible to arrive at reasonable maximum and minimum estimates of the number of NCSS from the project requirements, that the development time is probably already known to the customer (in the form of market deadlines for the project), and that therefore the model can be used to

plan the implementation in the available time (or show it to be infeasible). In the trade-off between staffing levels and development time the model exhibits behaviour similar to many other cost models in that it shows that reduction in development time is an expensive option causing a disproportionate rise in staffing levels and costs.

Comments on Putnam/SLIM

The premise that it is feasible to estimate the NCSS required to construct a software artefact appears to be the chief weakness of the model. As Amos (1993) reported at the European Software Cost Modelling Meeting (1993), "the assumption that our project managers would be able to estimate the required number of lines of code was in fact not borne out by experience". The difficulties with source lines of code (or source statements) have now been thoroughly discussed (Kitchenham, 1991a; Jones, 1986), a major stumbling block being that there is no satisfactory and accepted definition of a source statement or line of code.

The dynamic behaviour of software projects noted by Putnam does offer useful insights into the relationship between project variables such as: time-scale, staffing levels, development time and environment. Because of this, the Putnam model may well be useful if the necessary relationships could be established with a different sizing element such as the Bang metric (DeMarco, 1982) or a function-based metric. The strength of the model, and its most useful insight into estimating practice, is its insistence that it is essential to calculate the environmental factor from a history of past projects conducted in the same organisation. As noted earlier, the impact of the development

environment on the effort required is now well accepted within the estimating community (Boehm 1984, DeMarco & Lister, 1985, Kitchenham, 1991b; Miyazaki & Mori, 1985).

2.5 COCOMO

This is a well-documented method that is easily applied and highly visible (McTavish, 1984). The method was developed by Boehm (1981, 1984) at TRW (a United States defence industry company) and is based on his analysis of some 63 projects. It predicts not only the overall software development effort required and the time for development, but also breaks these down over the various phases of the 'waterfall' life-cycle model, which was first presented by Royce (1970) and comprehensively described by Boehm (1981). COCOMO is based on an input of estimated thousands of Delivered Source Instructions (KDSI), and exists in three forms:

Basic

The basic form of COCOMO provides only coarse estimates of work effort and development time without considering any modifiers to allow for project or environmental factors.

Intermediate

The intermediate model takes the basic effort estimate and refines it by taking into account 15 'cost drivers', i.e. project and environmental factors which are considered to have a significant influence on the work effort required to complete a project.

Detailed

Both the basic and intermediate models within COCOMO may be regarded as top-down, in that an overall estimate of work effort is first

calculated and then broken down into estimates for each individual work phase. However, advanced COCOMO is a bottom-up technique in that it proceeds by deriving effort estimates for each of the individual phases directly, which are then summed to give an overall figure. This is the most detailed of the three forms of the COCOMO model.

All three forms of the COCOMO model divide projects into one of three types, which are defined by Boehm as follows:

Organic

These are projects developed by small teams in a highly familiar, in-house environment. The development environment is stable with very little concurrent development of new hardware or operational procedures. There will be a minimal requirement for innovative data processing algorithms or architectures. There is a relatively low cost penalty if completion is delayed. Finally projects are not large, with few exceeding 50 KDSI of new software.

Semi-detached

The semi-detached mode of software development represents an 'intermediate' stage between the organic and the embedded mode development (see below). In this sense 'intermediate' may mean a mixture of organic and embedded mode characteristics or an 'intermediate' level of project characteristics. Boehm (1981) gives the following examples of 'intermediate' as it applies to the characteristic of experience with related software systems:

- The team members all have an 'intermediate' level of experience with related systems; Boehm (**ibid.**) suggests experience of

approximately one year for programmers and three years for analysts.

- The team has a wide mixture of experienced and inexperienced people; Boehm (**ibid.**) proposes three years or greater for experienced programmers and less than four months for inexperienced, for analysts the figures are six years and one year respectively.

- The team members have experience related to some aspects of the system but not to others.

In his use of the phrases 'experienced' and 'inexperienced' Boehm is referring to the "experience in working with related software systems".

Embedded

Embedded mode projects are characterised by a need to operate within tight constraints. The product must operate within (is embedded in) a strongly coupled complex of hardware and software, regulations and operational procedures. Boehm cites an electronic fund transfer system or air traffic control system as being examples of embedded mode projects. In general, an embedded mode project will be one in which the cost of changing other parts of the complex are so high that they may be considered *unchangeable*. The new software is therefore constrained to operate within the specifications of the other elements of the complex.

The basic, intermediate and advanced forms of COCOMO all rely upon an empirically derived model of software development of the form:

$$MM = c\,(KDSI)^k$$

where:

MM = man months (152 work hours representing a month in the COCOMO model)

c = a constant determined by both the type of the project and the form of the COCOMO model (basic, intermediate or detailed).

k = a constant determined by the type of the project.

In Boehm's development of COCOMO, he found that the basic model predicted effort within a "factor of 1.3 of the actual only 29% of the time and within a factor of 2 only 60% of the time" (Boehm, 1981). The refined versions of COCOMO achieve more acceptable results, intermediate mode estimates being within 20% of the actual 68% of the time, and advanced mode estimates within 20% of the actual 70% of the time, which Boehm contends is adequate for most estimating tasks.

Comments on COCOMO

As noted earlier COCOMO has the advantage of being extremely well documented and open to the researcher (McTavish, 1984). However the method has the disadvantage that it is based upon an estimate of Delivered Source Instructions. Superficially, COCOMO appears to provide a method that does not require calibration based upon previous projects, in that the method provides a series of cost drivers that users may estimate for their own

environment. Boehm, however, considers that the accuracy of COCOMO with respect to the TRW database is "probably somewhat better" than its accuracy to the general population of software projects and contends that the method should be calibrated, for which he provides guidance. This conjecture is borne out in the literature, and a number of researchers have suggested re-calibrations of COCOMO to suit the different development environments that they encountered (Miyazaki & Mori, 1985, Marouane & Mili, 1989). Unfortunately this point is missed by many practitioners, and the method is often used uncalibrated with disappointing results (Kemerer, 1987). Indeed, Londeix (1987) suggests a mapping between the COCOMO cost drivers and the environmental factors of the Putnam method. This has the effect of defeating the "previous project" basis that the Putnam method uses to derive environmental factors, and it also makes the invalid assumption that COCOMO can be used without calibration.

2.6 Function-based Methods

Boehm's (1981) review of software cost estimation methods included only one function based method — the GRC model (Carriere & Thibodeue, 1979). A survey of the literature revealed that a number of other function-based methods have been developed. These include Function Point Analysis (FPA) that was developed by Albrecht, originally as a productivity measure (Albrecht, 1979), and then later proposed as a method for estimating effort (Albrecht & Gaffney, 1983). A revision of the method has been put forward by Symons (1988). Other function-based methods are the 'Bang' metric (DeMarco, 1982) and ESTIMACS (Ruben, 1983 and 1985). FPA is the most commonly used of these

methods, having gained a wide degree of acceptance within the data processing industry. The Albrecht & Gaffney method has become a *de facto* standard within the information systems industry in the United States and in many European countries. Several large scale studies of systems development productivity have been carried out (Kunkler, 1985; IFPUG, 1988). Recently, in 1988, the United Kingdom Government's Central Communications and Telecommunications Agency adopted the Symons' variant as its official method for all government organisations and contractors. Additionally, real-time variants of FPA have been proposed (Jones, 1988; Reifer, 1987). An International Function Point User Group (IFPUG) has been established to promote the method, standardise the techniques used, and share experiences. Various national function point user groups have also been formed, and there is now a move to bring these various groups under one 'umbrella' organisation.

Function Point Analysis: Albrecht & Gaffney

FPA is based on Functional Size Measurement (FSM) which is defined as "measurement of the size of software by quantifying the functional user requirements manifested by the software" (ISO 1994). The advantages of a technique using an FSM are that it is:

- based on a conceptual and external view
- independent of development technology
- applicable at any life-cycle stage after requirements definition
- expressed as a numeric functional size index through the assessment of basic logical components.

The development of functional size measures was driven by the need to find a metric that is not dependent on:

- the *inputs* to the software development process (e.g., effort in the form of work-time, funds, skills, and elapsed time)
- the development technology that is used to derive a solution (e.g., methods and techniques, hardware and software tools)
- the operational technology used to deliver the solution (e.g. the hardware platform and the software infrastructure).

An FSM specifically aims to measure software size in terms:

- of the outputs delivered to clients
- that can be meaningfully understood by users of the software
- that can be interpreted in business terms
- that can be applied in a timely way, when metrics are required, both early in the software life-cycle and throughout the development process.

FPA is a method that provides estimates of the overall work effort; originally it also provided an estimate of the expected SLOC. FPA is based on the metric *function points* devised by Albrecht (1979). Function Points are a synthetic metric derived from counts of system attributes such as the number of input transaction types and the number of unique reports to be generated. Each of the system attributes has also to be classified as *simple*, *average* or *complex* FPA provides guidelines to assist the estimator in the task of complexity classification (Dreger, 1989; IFPUG, 1992; ISQAA Metrics Handbook).

Albrecht believes that Function Points (FP) provide several advantages over SLOC. Firstly, they can be calculated from system attributes known at an early stage in product development. An FP size measure can be obtained during the requirements definition stage which is a desirable property of a metric that is to be used for effort estimation. Secondly, it is claimed that Function Points can be derived by a relatively non-technical member of the project team. Lastly,

Albrecht implies that Function Points are independent of the technology used and therefore avoid the complications of language and implementation differences (this would provide a basis for comparing the productivity of differing development techniques, environments and tools, though this is not the subject of this research)

There are two basic steps in calculating Function Points. Firstly, FPA determines the Function Counts (FC) from five 'user function' categories: *external input types*, *external output types*, *logical internal file types*, *external interface file types* and *external inquiry types*. The method then adjusts the initial FC estimate of system size by an amount dependent upon certain 'processing complexity' factors, which are assessed to provide a Processing Complexity Adjustment (PCA). Albrecht & Gaffney provide a list of 14 processing complexity factors that are required to be taken into account. These factors are rated on a scale of 'degrees of influence' (DI) from 0 to 5 depending upon the amount of influence they are judged to have on the design (Table 2.4). The use of the PCA allows an adjustment of ± 35% to be made to the original FC figure. Thus, the adjusted function point size of the system is determined by the formula:

$FP = FC \times PCA$ where:

$PCA = 0.65 + 0.01 \times TDI$ where:

TDI is the total degrees of influence

The original version of FPA also provided various formulas to predict the expected SLOC and work hours for the project.

Table 2.4 Processing Complexity Factors			
1.	data communications	8.	on-line update
2.	distributed data or processing	9.	complex processing
3	performance objectives	10	reusability
4	heavily-used configuration	11	conversion and installation ease
5	transaction rate	12	operational ease
6.	on-line data entry	13	multiple-site usage
7.	end user efficiency	14	facilitate change

In their paper, Albrecht and Gaffney (1983) provide formulas for the prediction of both effort and SLOC for projects developed using the languages COBOL and PL/1. Work has been carried out by other authors and various tables of SLOC per function point are now available (Reifer, 1987; Jones, 1988; Dreger, 1989; ISQAA Metrics Handbook) for a wide range of languages, though with some minor differences. A more fundamental change to the original method is that the attempt to predict effort by the use of a set of given formulas has been abandoned (Dreger, 1989; IFPUG, 1992). The current technique is well described by Dreger (1989) and essentially consists of calculating productivity per function point for the development environment and then using this figure to predict the likely effort for new projects in the same environment.

Comments on the Albrecht & Gaffney FPA Method

Several problems arise when attempting to use this method, which relate to the concept of 'logical internal files', the classifications of 'simple' 'complex' and 'average', and also to the processing complexity factors. The concept of a logical internal file is difficult to apply and has to be supported by highly detailed counting rules that are provided in the descriptions of the method (Dreger 1989; IFPUG, 1992). A logical internal file does not easily relate to the entities and relationships used by modern systems development methods and database techniques, causing considerable difficulty for the counting practitioner. The method of rating various attributes as simple, average or complex is criticised by Symons(1988) as being too limited for example, an input of 10 data elements is rated as complex and so is an input of 100 data elements. The method of arriving at an estimate of processing complexity is subjective, and the distinction between some of the factors is not clear (for example 'design for performance' and 'design for a high transaction system') (Symons, 1988). Symons also criticises the factors for being equally weighted and the limitation on the number of factors (only 14).

Mk II FPA: Symons

Symons (1988) proposes a revised approach to FPA that attempts to overcome certain difficulties associated with the original method. The difficulties Symons identifies with the Albrecht & Gaffney method are:

- The basic logical components are difficult to identify in modern systems development practice; for example, how does a logical file relate to a database environment?

- A similar difficulty occurs with 'on-line' interactive transactions where input on a screen is followed by output on the same screen; is the screen input, output, or both?

- Symons also queries the weights given to different component types.

- He also criticises the choice of factors and weights for the general application characteristics given in the PCA component of the method.

One main objective of the MKII method is to view systems in terms that users "feel more comfortable with". The essence of his revision is as follows:

- A system is regarded as a number of logical transaction types, a logical transaction being any input → processing → output combination (as seen by users).

- An external interface (to another application) is treated as just another input or output, with the additional complexity of dealing with such an interface being compensated for in the processing complexity step.

- external enquiries are treated simply as logical transactions.

- Logical internal files, and attendant problems of interpretation, are abandoned in favour of the database notion of an 'entity'.

Symons considers that complexity classification of inputs and outputs should be dispensed with, and their size (in terms of the number of data element types) measured instead. For the processing component of a transaction, a reasonable size indicator is the number of entity types referenced by the transaction. Symons states that the calculation of Unadjusted Function Points, which corresponds to the FC calculation of Albrecht & Gaffney, should explicitly relate the weights of the components to the effort required to construct a system, rather than to the value as perceived by users.

Symons also proposes a modification to the PCA which he calls Technical Complexity factor (TCF). Symons, despite his own criticisms of PCA, feels that "the basic concept seems sound". Therefore, the Symons proposal retains the PCA step and extends the number of factors to 19, with the possibility of additional factors the user wishes to define, "if it is really justified". The additional predefined factors are given in Table 2.5.

Table 2.5 Symons' Additional Factors

15.	Requirements of other applications
16.	Security, privacy, auditability
17.	User training needs
18.	Direct use by third parties
19.	Documentation

Having extended the factors, Symons makes an alteration to the formula which now becomes:

$$TCF = 0.65 + C \times TDI$$

where C is a coefficient obtained by calibration.

Symons, having conducted a calibration of the coefficient C over several projects, finds that the 'best fit' value is 0.005 rather than the 0.01 of Albrecht & Gaffney.

Comments on MKII Function Points

The MKII method proposed by Symons is relatively easy to use and understand. It reduces the subjectivity associated with the original method, though there are still some problems when attempting to work from very early life-cycle products. The method lends itself to automatic calculation from CASE tool data. The one puzzling fact is that having provided a comprehensive critique of the PCA element of the original method, Symons decides to adopt it, albeit in modified form. In practice, the calibration coefficient recommended by Symons (C = 0.005) means that an adjusted function point figure will almost always be less than an unadjusted figure, as can be demonstrated if we assume that all 19 factors are weighted at 5:

TDI	=	19 x 5
	=	95
TCF	=	0.065 + 0.005 x 95
	=	1.125

Thus, the variation available using MKII weightings is from -35% to +12.5%. Since it is reasonable to assume that on most occasions the rating of the individual factors is unlikely to be 5, the TCF will tend to reduce the amount of effort required. This change to the coefficient has considerably skewed the variation available (± 35% for the Albrecht & Gaffney method) in a manner that seems counter-intuitive. If the idea behind the weightings of the components that make up the technical complexity adjustment is to relate them to the effort required to construct a system, then it seems reasonable to assume that for most systems the adjustment factor should be close to 1 (a total DI of 70 would be required for a TCF of 1) which is unlikely given the method of calculation proposed by Symons. A system that scored a mid rating for the total DI, would have a TCF of 0.89 thus:

95/2	=	47.5
TCF	=	0.65 + 0.005 x 47.5
	=	0.89

whereas with the Albrecht method it would have PCA of 1:

70/2	=	35
35 x 0.01	=	0.35
0.35 + .65	=	1

General Comments on Function Points

Several authors have noted that the Albrecht & Gaffney method is subjective and suggest it might be unreliable.

- "Opponents claim that the method requires some 'sleight of hand' in that computation is based on subjective judgement, rather than objective data..."(Pressman, 1987).

- An experiment carried out by Rudolph (1983) and reported by Low & Jeffrey (1990) found that, for a group of 20 GUIDE productivity project group members, estimating function points from a requirements specification produced a variation within 30% of the mean.

Other authors have noted that the various methods do not necessarily produce the same resulting size measure for the same system.

- "Variants in FP counting methodologies can result in variations of up to +/- 50%" (Jones, 1988).

- "If the methods of system sizing do not give sizes which correlate well ... is it perhaps because the measures do not measure the same 'size'?" (Symons, 1988).

The Low & Jeffrey comment is reported as a criticism of the method by several authors (Kemerer, 1990 and 1992, Pressman, 1990) and so has been included here in that context. However, it should also be noted that Low & Jeffrey report that "function point counts appear to be a more consistent *a priori* measure of software size than lines of code". They also report several counting practice variations among their analysts, several of whom appear to have made counting decisions in conflict with the guidelines being used. Guidelines not withstanding, there is difficulty in interpretation when rating the PCA factors. However, recent authors have conducted experiments to establish the reliability of function points and they have found the following:

- The United Kingdom Inland Revenue reported that collection of function point data by independent counters produced no significant differences (Bettendge *et al.*, 1990).

- In a comparison between the Albrecht & Gaffney method and a Canadian variant, the two methods both produced reliable results at the 95% confidence level. Both methods also gave the same results for the same system again within the 95% confidence level (Kemerer, 1990 and 1992).

- A further experiment comparing the Albrecht & Gaffney method with MKII function points also found that both methods produced reliable results at the 95% confidence level. However, the two methods did not produce the same count for the same system, the MKII method producing

a significantly lower result at the 95% confidence level than Albrecht & Gaffney (Rollo & Steven, 1993).

- In an experiment to find a correspondence between the features of a process control application and function points, it was found that, in using three estimators to calculate function points independently, the measurement experience was consistent with that of Kemerer(1990) that function points were reliable (Mukhopadhyay & Kerke, 1992).

Symons (1991) claims that the MKII method was calibrated to the original at 1,000 function points and that at sizes above this it will produce a larger count. The Rollo & Steven experiment showed that for a system of 140 function points the MKII method produced a lower count. This is not necessarily in disagreement with Symons, the important result being that both the MKII and the original method have been shown to be reliable. However, it does indicate the possibility that there is a curvilinear relationship between the two function point counts computed for the same system. This curvilinearity is somewhat worrying as in classical measurement theory there must be a constant linear relationship between two scales if they are merely different measures of the same attribute (Fenton, 1991). The crossover indicated by Symons' contention that MKII function points gives a larger measure above 1000 and the Rollo & Steven experiment that shows a lower reading (at 140 FP) with respect to the figure obtained by the Albrecht & Gaffney method, implies that the two scales are measuring different attributes of a system. There is therefore a need for further experimentation to establish whether or not this is the case.

A further criticism of function points is that they are not technology independent as was claimed by Albrecht (Verner et al; 1989; Ratcliff & Rollo, 1990). Symons in the MKII method explicitly acknowledges this and uses the analogy of the retail price index to justify the need to update the weighting factors used in MKII, thus allowing for changes in information systems development technology.

2.7 A Comparison of COCOMO and FPA

COCOMO and FPA differ in that the former is based on a prior estimate of SLOC, whereas the latter can be used to provide an estimate of SLOC from a given set of requirements. Both methods estimate the overall work effort that will be required by a project, though COCOMO goes further by providing a detailed breakdown of the effort estimates for each of the life-cycle phases.

The Function Points metric used by FPA is a measure of the functionality specified for the required system. It is possible to calculate this metric at an early stage in the project life-cycle once the requirements definition is complete. This makes FPA the more attractive method, as the SLOC figure input to COCOMO can be no more than at best an expert estimate until the project is virtually complete.

Since FPA provides an estimate of SLOC, and COCOMO uses a SLOC figure as an input, it might appear that combining the two models could provide an

effective method for cost estimation. However, two difficulties arise with this approach

- COCOMO considers modifiers to be cost drivers in that they influence the *effort* required to produce a system of a given size, whilst in FPA they are considered as complexity factors, the effect of which is to increase or decrease the *size* of the system produced.

- Several of the COCOMO cost drivers and the FPA complexity factors address the same area (e.g. execution time constraint and heavily-used configuration, which are both defined in terms of the amount of computer time available for the application under consideration); thus, they would modify the estimate twice in an end-to-end FPA/COCOMO combination.

Before any hybrid method involving FPA and COCOMO can be further developed, it will be necessary to resolve these questions via further empirical studies. With regard to JSD, therefore, a more fruitful long term objective may be to develop an entirely new estimation model for JSD, based on direct input of function point size calculated by an FPA-like method, or a sizing technique which allows a reliable estimate of the eventual SLOC size. This new model would be designed so as to provide a detailed breakdown of effort that is directly related to the development stages of JSD, rather than to a waterfall structure.

2.8 Conclusion

Having reviewed the state of current estimating techniques, there is no reason at this stage for discounting any of the techniques that have been discussed. The techniques that will be explored in the context of a JSD development are COCOMO and both variants of Function Points. The reasoning for this decision is based on the desire to provide both a top-down and bottom-up method tailored to JSD, in order to provide the best accuracy at all life-cycle stages. COCOMO is by far the most popular of the SLOC-based methods and has been favourably reviewed by several authors (Abdel Hamid & Madnick, 1986, Masters, 1985, Maroune & Mili, 1989). Both function point methods will be examined as, while the Albrecht & Gaffney method is well established, the MKII method has become a standard in the United Kingdom, and appears to address many of the reservations held about the original method, albeit introducing some of its own.

Chapter 3

JSD and Cost Estimation

Introduction

The purpose of this chapter is to examine the mapping of JSD project characteristics onto the candidate methods discussed in Chapter 2. In so doing, the difficulties of these mappings are explored. A proposal is made for a technique which will provide the JSD analyst with a method of applying FPA using the vocabulary of a JSD project. Finally, conclusions are drawn regarding the need for further work in the area of cost estimation for JSD projects.

3.1 JSD and COCOMO

As previously described, COCOMO has three modes: *basic*, *intermediate* and *detailed*. The results of applying the COCOMO method to a hypothetical JSD case study are given in Appendix A. The intermediate mode is described below together with a brief summary of the detailed mode, basic mode is not described here as it consists of steps 1, 3, and 4 of the intermediate mode COCOMO, which comprises four steps:

1. Calculate the initial effort estimate using one of the three formulas provided by the method, depending on the mode of project.

2 Adjust the initial effort estimate by assessing the effort multiplier values assigned to the various cost drivers using the guidelines provided. Now calculate the combined effort multiplier value in order to modify the initial effort estimate.

3 Using the appropriate formula, calculate the overall development time based on the modified effort estimate derived from the previous step.

4 Consult tables that give phased effort and development time distributions as a percentage of total project effort and development time. By using these percentage breakdowns, calculate the work effort, development time, and personnel requirements for the individual project phases.

Intermediate mode COCOMO as described above is applied at the product, or system, level. However, this mode can also be applied at the component level in which case the same steps are involved. Separate estimates are made for each major component in the system and the total estimate is then made up by summing all the component level estimates. The advantage of component level estimating is that often different components in a system have very different cost driver factors. It is for example possible that some components of a system are quite complex, from the processing viewpoint, while others are relatively straightforward.

Advanced, or detailed, mode COCOMO seeks to address limitations which Boehm has identified in the intermediate mode model. These limitations concern estimates for very large projects and are:

- "Its estimated distribution of effort by phase may be inaccurate;

- "It can be very cumbersome to use on a product with many components." (Boehm, 1981)

The Detailed mode COCOMO provides two main capabilities which address these limitations, these being:

- In intermediate mode COCOMO the phase distribution of effort is calculated solely on the basis of the size of the product. Detailed mode COCOMO provides phase sensitive effort multipliers because, in operation, it has been found that factors such as required reliability, applications experience and interactive software development affect some phases of the development more than others. The effect of this is that a set of phase sensitive effort multipliers are needed to compensate for the bias that would otherwise be introduced. Detailed mode COCOMO provides such multipliers for each cost driver attribute and these are used to determine the effort required to complete each phase.

- In the intermediate mode model, separate cost driver ratings can be arrived at for each component. However this process can be tedious and is unnecessarily repetitive if a number of components are grouped into a subsystem with practically all the same cost driver ratings. The detailed model avoids this problem by providing a three-level hierarchy in which:
  - ◊ Some effects which vary with each bottom level module are treated at the module level
  - ◊ Some effects which vary at the subsystem level are treated at that level

◊ Finally effects which affect the whole system are treated at the system level, an example being the effect of total product size.

In addition, the detailed COCOMO model has procedures for adjusting the phase distribution of the development schedule. For most other calculations the detailed model uses the same techniques as for the intermediate model. A full description of this model and its adaptation to suit JSD will be given in Chapter 5 where bottom-up estimation for JSD is considered.

No interpretation is required when making a COCOMO estimate with a JSD project, since all that is required is an estimate of the project SLOC and an assessment of the cost driver values. There is however a question as to the predictive reliability of a COCOMO estimate, it is reasonable to expect an increase in productivity, both in the specification and the implementation phases, given the high degree of case tool use in a typical JSD environment, therefore the cost drivers concerned with case tool utilisation may require calibration. The code generation phase in particular is likely to be much more productive than for conventional hand coding. Apart from manual intervention for program inversion and data declarations, the task of generating program code is limited to writing simple sequential statements associated with Jackson Structured Programming (JSP) 'action boxes'; all control code is produced automatically. Given the difficulties with reliably estimating the SLOC size of a delivered system early in the life-cycle, it would be more useful to the JSD project estimator if a variant of COCOMO, based on a metric related to the JSP structure diagrams, could be developed for use as a bottom-up method.

Having made the effort estimate using COCOMO, there is then a problem of interpreting the effort breakdown in the light of a JSD environment. The software project phase models and associated percentage breakdowns used by COCOMO do not relate to the phase breakdowns of JSD projects since the latter do not follow a conventional life-cycle model it should be noted that the COCOMO phases are a subset of the full waterfall life-cycle model. It is necessary, therefore, to develop software project phase breakdown figures particular to JSD before COCOMO can be fully applicable to JSD projects. In other words life-cycle phases of JSD (modelling, network and implementation) have to be mapped onto the life-cycle phases of COCOMO. These phases and the percentage effort required by each phase for small, medium and large organic mode projects is given in Table 3.1.

Table 3.1. Percentage effort by COCOMO development phase

Phase	Small	Medium	Large
Plans & Requirements	6	6	6
Product Design	16	16	16
Detailed Design	26	24	23
Code & Unit Test	42	38	36
Integration and Test	16	22	25

By examination of several projects developed using JSD, and attempting to map JSD project phases onto those in Table 3.1, gives the results shown in Table 3.2 were obtained. This data, which only applies to large projects (in excess of 128 KDSI), is extracted from Butcher & Laddington (1987). Unfortunately, the

authors do not state how they mapped the JSD stages onto conventional phases. Examination of Tables 3.1 and 3.2 reveals that where the Plans & Requirements phase in the conventional COCOMO model consumes some 6% of the effort available, in the JSD equivalent this phases require 14% of the effort, the design phase has increased from 16% to 39% of the total effort.

Table 3.2. Percentage effort in the JSD life-cycle mapped onto COCOMO phases

Phase	Large
Plans & Requirements	14
Product Design	39
Detailed Design	16
Code & Test	28
Integration & Test	17

In the later JSD project phases of Table 3.2, effort in JSD projects has been reduced by 10% to 12% over the conventional COCOMO model phases. It will be noted that in both tables the total percentage figure adds up to more than 100%. This is because the COCOMO model assumes that the Plans & Requirements phase is completed before any estimate can be made, hence, 100% of the effort being estimated is for those project phases that follow on from Plans & Requirements.

Examining projects from Michael Jackson Ltd (Sharelink) and from Abbey National (Pensions), it has been found that the percentage effort per COCOMO software project phase, accepting the difficulty of performing this mapping, is

similar to the figures obtained by Butcher & Laddington, but with a slightly smaller figure for the detailed design stage. There being no agreed mapping, it seems better to devise percentage effort figures based around a JSD project life-cycle rather than attempting to map to a more conventional life-cycle view of software development From the data available the JSD specific phase breakdown has been calculated, this breakdown based on the stages given in the JSD Manual (LBMS, 1991) is detailed in Table 3.3

Table 3.3 JSD Effort by Stage

JSD Stage	Percentage Effort
Project Initiation	-
Architecture (AR)	11
Network (NS)	28
Physical Design(PD)	16
Construction (CO)	28
Integration (IN)	17

3.2 JSD and Albrecht & Gaffney Function Points

As already described in Chapter 2, the Albrecht & Gaffney function point method has two main stages: firstly the calculation of the Function Count (FC), and secondly the assessment of Processing Complexity Adjustment (PCA).

FC Calculation

The FC parameter categories are as follows, with each being given a brief description based on Albrecht & Gaffney:

External Inputs

Each unique user data or control input type that enters the boundary of the system.

External Outputs

Each unique user data or control output type crossing the system boundary.

Logical Internal Files

Each major logical group of user data or control information within the system.

External Interface Files

Each file passed or shared between applications.

External Inquiries

Each unique input/output combination, where the input causes an *immediate* output to be generated.

These definitions and the terms contained therein (such as 'file' or 'group of user data') require careful interpretation. The definitions and correspondences that were first constructed as mappings between FPA and JSD are given below. These were constructed by the analysis of a JSD case study specification and subsequently by conducting a function point analysis of a completed commercial project (Rollo & Ratcliff; 1988; Ratcliff & Rollo, 1990), which played a significant part in the derivation. The result of applying this mapping of the Albrecht & Gaffney method to a hypothetical case study is presented in Appendix A and to an actual project in Appendix B.

External Inputs

Each action of each entity is deemed to represent a unique external input type. Also included in this category are inquiry inputs that do not require immediate responses (i.e., slower than an on-line user inquiry where response should be no more than a few seconds). JSD actions are the means by which external 'real-world' entities communicate with their 'model' processes.

External Outputs

In this category are placed output data streams generated by functions in response to regular real-world time intervals (i.e., output triggered by 'time grain marker' streams in a JSD network). Also included in this category are:

- outputs generated automatically (by 'embedded functions' in a JSD network);
- slow-response (non-immediate) outputs.

There is no real difficulty of interpretation here. Slow-response outputs are included because, while they are generated by a user inquiry, they do not form part of the FPA definition that inquiries generate *immediate* responses.

Logical Internal Files

These are equated with the data (state vector) accesses that would be associated with the various functions listed in the requirements. The requirement to count "logical files as seen from the user view" caused

some difficulty of interpretation, due partly to the need to infer the user view from a completed technical specification. The decision was taken that, as the user will have specified the functions in the system, the associated data accesses will adequately reflect the user's view of the logical data files. The implication of this is that data accesses of functions provided, but not specified by the user (e.g., for system 'housekeeping'), should not be included in the count.

External Interface Files

Within a JSD network, it is possible to view subsets of the network as being associated with each of the functions required by the user. Therefore, the decision was taken to regard functions as separate *applications*, in which case each different kind of entity is equivalent to an external interface file. This is *not* the same as for the logical internal files, as we are now viewing (the set of state vectors for) each entity class as a file *shared* by applications.

External Inquiries

Functions that are required to give an immediate response after being triggered by an incoming request/inquiry are counted as external inquiries. This follows Albrecht & Gaffney exactly.

The following should be noted:

- The time grain marker streams which generate regular periodic output are not considered to contribute to any of the above input categories; from a user view, time grain markers and the functions

they trigger are automatic and do not require any inputs to initiate them

- The previous point raises a more general issue. The above mapping between JSD and function points will not take into account every single process and data stream that may eventually appear in a JSD network. Nor should it — only that which is determined from the user-oriented requirement's specification is relevant, not what is produced as a consequence of using the JSD method itself.
- There is one proviso to the last point — the analysis *does* depend on some initial modelling having been accomplished. This tentatively identifies entities and their actions, their states and also what state vector accesses will eventually appear in the final specification to satisfy the functional requirements; a first estimate of effort and cost can thus be obtained. Later, when the detailed real-world modelling has been completed and validated, revised (and presumably more accurate) estimates can be computed.

Processing Complexity Adjustment

There are no difficulties in interpreting the processing complexity factors in the context of a JSD specification, though the assessment of the degree of influence of any particular factor is a subjective one on the part of the analyst. Both the ISQAA handbook and Dreger (1989) lay down a series of guidelines for assessing the Degree of Influence (DI) of processing complexity factors, and these reduce the subjectivity in the assessment. The factors to be considered are presented in Table 3.4. below:

Table 3.4. Factors affecting the overall processing complexity adjustment					
	Factor	**DI**		**Factor**	**DI**
1	data communications		8	on-line update	
2	distributed data or processing		9	complex processing	
3	performance objectives		10	reusability	
4	heavily-used configuration		11	conversion and installation ease	
5	transaction rate		12	operational ease	
6	on-line data entry		13	multiple-site usage	
7	end user efficiency		14	facilitate change	

The interpretation of the DI values for use in Table 3.4. is:

Not present or no influence	0	Average influence	3
Insignificant influence	1	Significant influence	4
Moderate influence	2	Strong influence throughout	5

As a result of comments and discussion with practising estimators after the first publication describing this mapping (Rollo & Ratcliff, 1988), the originally proposed treatment of external interface files as being part of the communication between JSD functions and processes (namely the state vectors) was reviewed and, in light of the more detailed set of counting practice rules available in Dreger (1989) and the IFPUG counting practices manual (1992), was seen to be an incorrect interpretation of that intended by Albrecht & Gaffney. The modified proposal (Ratcliff & Rollo; 1990,) was that this should

be treated as being a communication between separate applications that is, separate to the one being considered.

3.3 Estimating in a JSD Environment - A set of Counting Rules

The new proposal, originally presented by Ratcliff & Rollo (1990), consists of a set of counting rules that are applied directly to a JSD specification using the vocabulary of a JSD analyst. The mapping between the FPA Function Count parameters and JSD attributes is presented below:

Counting rules

External Inputs

Actions of, i.e., action messages associated with, entities

State vector inspections of the states of external entities

Inquiry input data streams not requiring immediate responses ('immediate' in an on-line or interactive sense).

External Outputs

Output data streams generated at regular real-world time intervals by 'time grain marker' inputs (note that the latter themselves make no direct contribution to the estimation procedure)

Output data streams generated automatically without any input stimulus (e.g. by so-called 'embedded' functions)

Output data streams not required as immediate responses to inputs.

Logical Internal Files

Each state vector inspection by a function that was specified by the user.

External Interface Files

Essentially as Section 3.4, regarding 'application' as any system built separately from the one under consideration and where 'file' corresponds to a data stream or state vector inspection connecting the two systems.

External Inquiries

The messages associated with input and output data streams related to inquiry functions, requiring immediate response, in the JSD specification.

Complexity Rating

It will be recalled from Chapter 2 that Function Count estimation also involves allocating a 'simple', 'average' or 'complex' rating individually to all items in the five parameter classes. Guidelines for assisting this evaluation of complexity are provided by Albrecht & Gaffney and relate to such factors as the number of (logical internal) files an item makes reference to, and/or the number of data element types it consists of. However, the guidelines are somewhat subjective. To overcome this subjectivity, and to keep matters of interpretation as straightforward as possible in relation to JSD, the following approach is proposed:

Simple	Less than M data element types
Average	Between M and N data element types inclusive
Complex	More than N data element types

For simplicity, it was further recommended that the values chosen for M and N (values of 6 and 10 respectively are suggested) should be the same for each of the five parameter classes. Note that for each complexity rating in each

parameter class, FPA provides a predetermined multiplicative weight factor that is intended to represent the 'value to users' of items of that particular class and complexity. The assessment of complexity rating described above is suggested as a provisional technique, but requires further consideration in relation to the technique of Symons (1988). The counting rules given above were used by Michael Jackson (UK) Ltd to estimate client projects (MJUK 1989b). The results of applying these rules to a commercial project are presented in Appendix B. An alternative set of ratings for JSD attributes was provided by Dedene (1989), these were not considered further, as the results of the MJUK experience had indicated that an alternative to the Albrecht & Gaffney method would be desirable.

Processing Complexity Adjustment

The processing complexity adjustment should be carried out in accordance with the guidelines given by Albrecht & Gaffney. Further guidance is given by Dreger (1989), the IFPUG counting practices manual (1992), and the ISQAA metrics handbook.

3.4 JSD and The MKII Function Points Method

The MKII method of FPA relies upon the identification of a series of transactions which are defined as input → process → output structures. Having identified these transactions, the analysis consists of counting all inputs and outputs as well as entity reference types, these being what Symons uses as an indicator of the amount of processing in a transaction. The results of applying FPA MKII to a commercial project are given in Appendix C. The description of the mapping between JSD and MKII FPA will start with Unadjusted Function Points and be followed by Technical Complexity Factors.

Unadjusted Function Points

The calculation of Unadjusted Function Points is the equivalent of Albrecht & Gaffney's FC calculation. The mapping onto JSD parameters was achieved as follows.

Entities

At first glance, there would seem to be no difficulty in counting entity reference types, since JSD modelling, in part, identifies entities. However, a JSD entity is a dynamic life-history model of some real world object, whereas 'entity' in the Mark II FPA sense is database oriented (Symons, 1988) and can therefore apply to static data objects in the real world as well as the dynamic objects that are the subject of JSD entity life history analysis. The effect of this difference will thus be that JSD modelling will normally generate fewer entity types than conventional entity analysis. As a consequence, applying Mark II FPA but using the JSD concept of an entity is likely to result in a count of Unadjusted Function Points less than would otherwise be obtained. Despite these reservations, it was decided that references to a JSD entity (i.e., accesses to the entity state vectors) would be counted as the metric for this category.

Transactions

During the modelling stage of a JSD project, there is no direct equivalent of a transaction. Nevertheless, entity actions generate input and result in a change to the state of the system; thus they are part of a transaction, though not as originally defined in MKII FPA (Symons, 1988). Transactions that fit the Symons definition begin to appear at the specification stage when system functions are considered. Such structures

generally correspond to inquiry functions, and here there is no problem of interpretation. However, there are many inputs, and outputs, involved in a JSD specification that are not part of a transaction. Nevertheless, since they are associated with processing taking place, they cannot be ignored and so were counted within the input or output categories.

Technical Complexity Factors

Symons' Technical Complexity Adjustment is the equivalent of Albrecht & Gaffney's Processing Complexity Adjustment. Symons includes 6 additional complexity factors to the 14 of Albrecht & Gaffney (see Table 2.5 in Section 2.6).

Comments on MKII FPA

Since the initial appraisal of MKII FPA, Symons has published a fuller description of the method (Symons, 1991), in which he has revised the description of an entity, and clarified his description of a transaction. The revised description of an entity brings it in line with the 'major entity' as described in SSADM (NCC, 1990). An SSADM major entity and the JSD notion of an entity coincide more closely than the database concept of 'entity', and the reservation with regard to counting JSD entities has therefore been eliminated. In reconsidering the MKII method in the light of Symons (ibid.), it was considered that the difficulty with the concept of 'transaction' has been largely overcome as Symons now regards the operations of create, update and delete as transactions. The only problem that remains is that the full range of transactions in the system will not have been defined until the end of the network stage of JSD. However, in two experiments conducted with groups of students one of which was published in Rollo & Steven (1993), no difficulty

was experienced in reliably estimating the number of entity references (state vector accesses) likely to be required from an early life-cycle description of the functional requirements of the system.

This reappraisal means that the MKII method is now considered to be a good match with the requirements of a JSD analyst. The method works well with data rich applications (CCTA, 1991; Rule, 1991; Sheppard & Turner, 1993). The method however fares poorly when it is used in the context of real-time or scientific systems development (Rule, 1991, 1993; Sheppard & Turner, 1993; Rollo, 1994). It should also be noted that in his original paper, Symons did not publish any method for converting the function point counts into estimates of effort, thus, the formula provided by Albrecht & Gaffney was used for calculating effort. This is justified by Symons' claim that the MKII method had been calibrated to give the same count as the Albrecht & Gaffney method at 1000 FP. Symons (1991) has proposed that the method for calculating effort from the adjusted function point counts should be by determining, from past projects if they are available, the productivity (in function points per hour) of the local development environment and using this figure calculate an estimate of effort. In the event that sufficient past projects are unavailable to determine productivity, Symons publishes a set of industry standard figures which may be used until sufficient local data has been accumulated.

3.5 Conclusions

This chapter has examined the candidate methods: COCOMO, Albrecht & Gaffney FPA, and finally the MKII FPA, as potential methods to be employed for estimating effort in a JSD environment. Proposals have been made for a mapping between the parameters of Albrecht & Gaffney FPA and the characteristics of JSD developments. Additionally rules have been formulated for interpreting JSD attributes in order to allow a MKII FPA estimate to be prepared.

The work carried out in the various experiments and attempts to estimate past projects have shown that the function points method as described by Symons (1991) is suitable as an early life-cycle estimating method for a JSD project. The experience gained by Michael Jackson UK Ltd, which continued when absorbed into Learmonth and Burchett Management Systems, has led them to the same conclusion. The Albrecht & Gaffney mappings as detailed in this chapter have also been found to be suitable. For commercial use however, given the support by CCTA, it is recommended that the method that is adopted should be MKII FPA. The method is only suitable for data rich applications as characterised by commercial information systems, and performs poorly in the context of real-time or scientific applications. Real-Time and scientific applications are examined in detail in the next chapter.

Chapter 4

Top-Down Sizing and Estimating JSD Specifications

Introduction

The purpose of this chapter is to review the effectiveness of the top-down methods investigated in the context of a JSD environment and to propose a suitable new technique. The chapter will look at estimating methods in the context of real-time, scientific and other highly constrained systems, and discuss the difficulties that exist with current real-time and scientific adaptations of FPA, before proposing an extension to the function point techniques which will overcome the main difficulties identified.

4.1 Disadvantages of FPA in Real-Time, Scientific and Engineering Systems

As noted in Chapter 3, both the original Albrecht method and the MKII Symons variants of FPA produce reliable results in the data rich environment normally associated with commercial information systems. Both of these methods are suitable for top-down estimation of a JSD project provided that it is a data rich project and that, for the original FPA, the mappings provided in Chapter 3 are used. However in systems that are not data rich, such as real-time and scientific systems, these two methods do not perform as well. The general tendency is for

Function Points to give serious underestimates (Whitmire *et al*, 1992; Verner *et al*, 1989; Rule, 1993); there are also difficulties with commercial systems such as electronic fund transfer and point of sale systems (Rule, 1993). Various authors have suggested extensions to FPA to take account of the features of these systems (Jones, 1986, Reifer, 1987; Whitmire *et al*, 1992; Mukhopadhyay and Kekre, 1992, Rule, 1993).

In attempting to analyse why Function Points perform well in data rich commercial environments and poorly in the scientific, engineering and real-time environments, it is useful to consider the characteristics of software systems. DeMarco (1982) provides us with some insight in his discussion concerning the specification of a system. DeMarco argues that a system cannot be specified from a single perspective and asserts that a complete specification of a system must have three perspectives

- A *functional* model (a view of what the system does);
- A *retained data* model (a view of what the system remembers);
- A *state transition* model (a view of the different behavioural states that characterise the system)

He contends that these three components of the requirements model can be thought of as projections onto a different dimension or space in the problem domain — the function model onto *function space*, the retained data model onto *information space* and the state transition model onto *control space*.

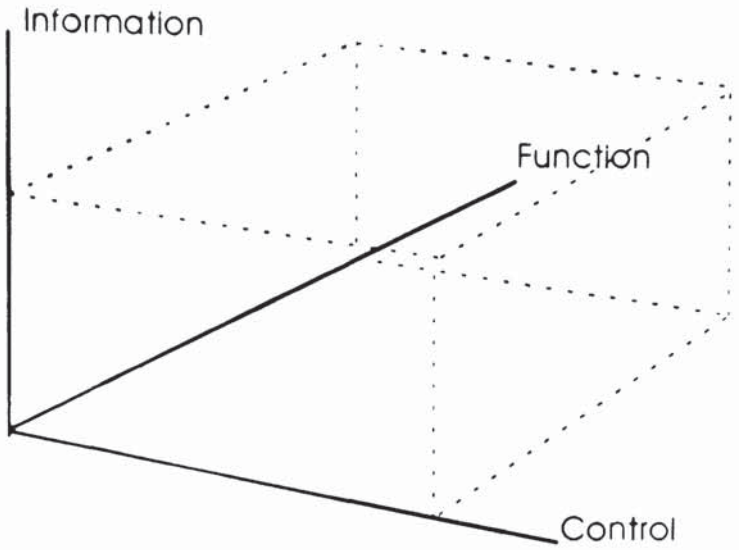


Figure. 4.1 DeMarco's Model of System Size

The total 'size' of a problem is thus made up by a contribution from each of the three dimensions, see Figure 4.1. This concept of three dimensions of the problem space is used in many of the structured systems development methods in current use (Yourdon, 1989; NCC, 1990).

Many systems have the characteristic that they are dominated by one particular dimension. An examination of the characteristics of such systems will help in explaining why FPA does not adequately deal with all classes of system.

- *Data Rich* (Information Systems)

  Information systems are, generally, data rich. That is, they are dominant in the information space dimension. This is not to say that they do not have elements of the other two dimensions, but that the dominant contribution to the problem size is from the information space and the

data interchange between the problem domain and the software system. That is, the data which crosses the system boundary. Such systems generally model or retain information about real-world entities, physical or otherwise. Their main purpose is to collect, store and distribute this information, commonly in restricted or controlled ways. A good example of such a system is the library case study that is illustrated in Appendix A.

- *Function Rich* (Scientific or Engineering Systems)

These are systems where the dominant contribution to the 'size' of the problem is the function or processing required to transform the input into the output. Scientific and engineering systems are required to model the processes or phenomena that operate on real-world entities. Consequently, they may hold little or no data about the real-world and what they do hold is often fixed information, such as size, shape, etc. Thus, in such a system, an aircraft may be represented by the semantic rules that govern its behaviour in some problem domain under investigation (e.g., aerodynamic) with very little data about the actual entity. These behavioural semantics are used to govern the computation of algorithms that transform inputs about wind velocity say into outputs that describe the behaviour of the aircraft in certain wind conditions.

- *Control Rich* (Real-Time and Embedded Systems)

Real-Time systems monitor and control real-world artefacts. A real-time system is required to control a process in a time scale that is determined

by the process and not the computer. Real-Time and embedded systems control devices that operate in the real-world, and they must do so in such a way that information is not missed or corrupted because the software system cannot keep up. They have problems of timing, concurrence and safety. A real-time system may have no user interface (other than a few buttons or switches) and little or no retained data. Some real-time systems do not retain or process data at all but merely monitor and control a series of devices, passing the data that they collect to other systems for processing or storage (Mellichamp, 1983). Consider a simple (relatively) embedded system to control a domestic washing machine. This system must monitor the important factors involved in washing clothes, that is the temperature, time, amount of detergent and such like. The user interface to the system will be the various switches that set the desired wash program. In a modern machine these switches are usually set to indicate the type of clothing being washed (e.g., soiled whites, woollens etc.). The system need store no data at all other than fixed look-up tables to tell it what values the various variables, such as wash time and water temperature, should have for the programme selected. The system must, however, monitor a variety of sensors in a sufficiently timely manner that it can direct the behaviour of devices such as the heater element, the wash/spin tub, water inlet, and drain outlet valves.

- *Hybrid Systems*

Another class of system is one that is a hybrid of all; or two, of the above types. Such systems will have equal or near equal dominance of two or

more of the dimensions proposed by DeMarco. The problem 'size' in a hybrid system is made up of roughly equivalent contributions from two or all three dimensions; that is; such a system may hold data about entities, as well as modelling their semantic behaviour and monitoring and controlling them in real-time. Examples of such systems are avionics systems and command and control systems (DeMarco, 1982).

Methods such as FPA do not address all three of the dimensions proposed by DeMarco. FPA is concerned almost exclusively with the single dimension of information space. The only concession FPA makes to the other dimensions is in allowing the processing complexity adjustment to vary the function point count. The amount of variation permitted is slight (±35% or -35% to +12.5% in MKII FPA), and not entirely due to factors that can be considered to be from either of the other two problem dimensions. For example, the factor 'heavily used configuration' is clearly referring to the target machine in the *solution* space and not to any dimension in the *problem* domain. The amount of variation allowed by either version of FPA is insufficient to adequately model an application which is function or control rich, or is a hybrid of any of the three dimensions.

The limitation of FPA, in not addressing all three dimensions of the problem space does not apply to SLOC-based estimating methods. These methods are addressing the solution space of the application and therefore make no allowance for the dimensions of the problem space. For this reason SLOC

methods are currently the most popular in the real-time and scientific application area of the software development industry. SLOC-based methods are, however, subject to the inherent difficulty of obtaining accurate estimates of the number of lines-of-code to be developed. This limits their usefulness in estimating effort for contract negotiation.

As mentioned earlier, several authors have proposed modifications or extensions to FPA in an attempt to overcome its weakness in addressing one or more of the areas discussed above. These modifications have been motivated by unsuccessful attempts to apply FPA in areas which are not characterised as data rich. The various authors have discovered, when attempting to apply FPA, that it does not adequately represent the size of the task of constructing a scientific or real-time system. These extensions will now be considered.

4.2 Feature Points

Feature Points were developed by Jones (1986) to address the problem of Function Points in applications other than data rich information systems projects. Jones takes the view that such applications are characterised by the complex internal processing they perform. Feature Points were derived from a modified form of Function Points that Jones had originally devised in 1985. Jones had experienced difficulties with Albrecht & Gaffney FPA. He felt that the counting rules were too complex, and that a simpler method of counting was required. His version of Function Points eliminated the need for classifying the complexity of each component, by simply using the average weights from Albrecht. Jones also revised the Processing Complexity Adjustment factor

(Albrecht's PCA) quite significantly. In his variant of FPA, the adjustment factor was calculated by rating, on a scale of 1 to 5, the overall complexity of the logic, the data, and the code. The adjustment factor is then derived by summing these ratings and then looking up the sum in a table, which provides the adjustment factor to be applied to the unadjusted function point counts. Jones' adjustment factor allowed for a variation of ± 40%. This technique was incorporated by Jones into his commercial tool SPQR/20.

Jones found that his modified function point method was unsatisfactory for various applications. In an experiment to size telephone switching software, Jones added a new feature, an algorithm parameter, to his function point definition. At the same time he reduced the weighting applied to internal files from 10 to 4 and allowed a weight of 4 for the algorithm parameter. This has the effect of regarding internal files as having the same contribution to problem size as inputs and algorithms. Jones defines an 'algorithm' as "the set of rules that must be completely expressed in order to solve a significant computational problem". Examples of such algorithms that Jones provides are a Julian date conversion routine and a square root function.

The feature point metric is an attempt to address dimensions other than simply data. However, this metric does not address the control dimension at all, and, while an algorithm is an element of function, not fully the function dimension. The amount function in a system is not solely due to the algorithms, as defined by Jones. A further difficulty is that the definition of an algorithm, and the

accompanying examples, are not adequate for dealing with counting in large systems -- after all even a straightforward data processing system will probably invoke a Julian date conversion routine. The main point here is that the algorithms as illustrated are far too low-level to account for the amount of function in a system. Function point analysis is a technique that can, arguably, be used early in the life-cycle, but it would be extremely difficult to count algorithms at this level of detail from a requirement's specification. In a JSD development, it would be possible to ascertain algorithms, though it is generally the case that they are devised, coded as functions and then regarded by the JSD practitioner as callable functions.

Symons (1991) argues that the estimation of an algorithm is not a simple matter of counting them and assessing their complexity in the same way one can for an input. In the case of a mathematical algorithm, it is either already known in which case a routine probably already exists and one should use that, or a new algorithm has to be devised by a mathematician, numerical analyst, or some other domain specialist. To predict how long it might take to devise a new algorithm is extremely problematic. Symons points out, that the formula $e = mc^2$ is trivial to program, "but who knows how long it took to devise". No method of software problem sizing can possibly account for the development of new algorithms. The task of estimating the development of an algorithm must remain the province of the domain specialists who are charged with its development. Software development on systems that are at the leading edge of computational or domain knowledge will never be well served by estimating

methods, except for those portions of systems that are not breaking new ground (Rollo, 1994).

4.3 Analytical Software Sizing and Estimating Technique ASSET

ASSET is a software sizing technique for real-time and scientific systems devised by Reifer (1987). ASSET uses a size metric known as Reifer Function Points. Reifer was motivated by the need to find a consistent and reliable method for estimating the final developed SLOC figure from information available at an early life-cycle point.

The Reifer formula for SLOC is given as

$$SLOC = (ARCH)(EXPF)(LANG \times FP_A - MV_S)^2$$

where:

ARCH is an architectural constant, whose value is determined from a table dependent on the type of system architecture thus: a centralised architecture scores 1; a fully distributed architecture scores 2.1; while an array processor scores 0.9. Reifer derived these scores empirically.

EXPF is calculated by multiplying a calibration constant by the sum of a set of multipliers, each derived from a table thus: requirements' volatility scores between 0.95 and 1.18; database size scores between 0.94 and 1.11; use of modern programming technique scores between 0.93 and 1.11; language experience scores 0.91 to 1.13. Reifer's article does not give

any indication as to how to determine the value within a given range. There is, however, a commercial tool called ASSET-R that presumably contains the necessary guidance.

LANG is an expansion factor for the target language, which can be looked up in a table or is contained within the commercial tool.

FP_A is calculated from a formula that depends on the type of system, i.e. function-rich or control-rich. For function-rich systems the formula is:

$$FP_A = N_{inputs} + N_{outputs} + N_{master\ files} + N_{modes} + N_{inquiries} + N_{interfaces}$$

The term 'modes' is not defined nor is it clear from the article what exactly is meant. The formula for FP_A does not weight the components, as Reifer asserts that his empirical data gave better results with no weighting. For control-rich applications, the formula is modified by the addition of two extra components. These are: *stimulus response* relationships and *rendezvous*, which are not defined though their meaning is clear.

MV_A is a normalised measure of the 'maths volume', the Reifer article gives no indication how it is to be calculated.

Difficulties with Reifer Function Points as a function-based metric are that, while modes are not defined, it would seem from the article that they are a characteristic of the solution and not of the problem space. Some, though not all, stimulus response relationships can be derived from the problem space and may therefore have validity in a function-based method which is intended to be oriented in the problem domain. Rendezvous however, are an element of the designed solution and therefore inappropriate in a function based approach. The counts of the factors that Reifer uses in the formula for FP_A will not be available early in the life-cycle so that the early estimates needed for contractual negotiations will have to be based on estimates.

4.4 3D Function Points

These metrics have been devised by the Boeing Company (Whitmire *et al*, 1991). The 3D Function Point metric recognises, and attempts to address, the three dimensions of the problem space, identified by DeMarco (1982). This is achieved by the addition of three components to the Albrecht & Gaffney components. The additional components are *transformations*, *states* and *transitions*. The definitions for these are:

> *Transformations:* "...is a set of process steps and the set of governing or constraining semantics to transform one set of input data into a set of output data." The authors are at pains to point out that the steps are *processing* steps, which they claim are determined from the problem space rather than the programming steps (which of course are from the

solution space) They do not give an example, but presumably the following fragment from a requirements specification would provide a suitable example

> Calculate weekly pay as the sum of standard rate hours times the standard rate plus the overtime hours multiplied by one and a half times the standard rate

This is a set of process steps and the rules governing their application that might well be found in a requirements specification, or indeed as the instructions to a clerk who is required to perform the task manually, the inputs being the number of hours worked at the standard rate and the number worked as overtime as well as the two rates of pay, the output being weekly pay. The 3D function point proposal contains rules to help count transformations in image processing and numerical problems. A table relating the number of steps in a transformation to the number of statements is provided to allow the estimator to assess the complexity of the transformations being counted (Table 4.1). The complexity is assessed as being Low, Medium or High.

States: "...A state is that set that contains one and only one value for each condition of interest in the problem." The states are counted directly from the finite state machine representation of the problem space. If it is described as a state transition diagram then this is a matter of merely counting the states from the diagram, if not then a state transition representation must be constructed. A consequence of this is that there is

no complexity rating required for a state or a transition as "a state can only represent one state in the problem — if any conditions in the problem are changed then we move to a new state" (Whitmire, **ibid**).

Table 4.1. Complexity Levels for Transformations in 3D Function Points

Statements	1-5	6-10	10+
Steps 1-10	L	L	A
11-20	L	A	H
21+	A	H	H

Transitions: These are defined as being a valid path ("that is defined within the problem space") from one state to another. The authors claim that there is at most one transition from any one state to any other (i.e., they do not consider co-determinism). As already noted, transitions have no complexity rating. However both states and transitions are weighted at 5.

No empirical evidence is offered to support the validity of the 3D Function Point metric, which the authors acknowledge such evidence is needed, and they propose the design of a validation experiment.

For the other components of Function Points the authors use the counting rules as given in the IFPUG Guide (IFPUG, 1988). However, when considering the matter of complexity adjustment, they take the view that the proposed metric now accounts for all three dimensions of the problem. Therefore, there is no need for adjustment by what they regard as subjective measures.

4.5 Function Points for Process Control Applications

A variant of function point sizing for process control applications has been developed by Mukhopadhyay & Kerke (1992), which uses the features particular to a process control system. Process control applications are required to control the facilities of the machines used in an automated manufacturing process. The features used are: the Communications Feature (CF), the Position Feature (PF), and the Motion Feature (MF). The authors devised a model for mapping between these counts and Function Points. They have also developed a technique for estimating the expected delivered source instructions in order to use COCOMO. They report that they achieved a very high degree of agreement between the estimated function counts and the actual function counts (the actual figures were calculated by three analysts independently). Having found a reliable mapping between their features and Function Points, the authors went on to develop a model to estimate effort directly from the features of the application.

4.6 Function Points for Highly Constrained Systems

The European Function Point User Group (EFPUG) was formed in 1990 (it was later renamed to the United Kingdom Function Point User Group (UFPUG)), to pursue the use and development of MKII Function Points. One of the Special Interest Groups (SIG) of EFPUG is concerned with function point extensions, that is the extension of FP into areas that it does not currently cover. The SIG has been examining the possibility of devising a function point extension to cover what it calls 'highly constrained systems'. These are defined as: "... one for which user requirements include specified performance tolerances for any of a set of dimensions, including *accuracy, reliability, availability, usability, time behaviour*..." They list the following applications as exhibiting the characteristics of a highly constrained system:

- Embedded systems
- Telecommunications systems
- Systems or system-integration software
- Command and control systems
- Electronic funds transfer and point of sale systems

The list is not considered to be exhaustive but merely illustrative.

The method that they propose views a system as a set of layers of functionality, which derives from considerations of abstraction and decomposition. This view supports modern concepts of information hiding and coupling which is now

considered good practice in the Software Engineering field. Each layer is the equivalent of a particular level of abstraction, thus, we may have a layer dealing with the human interface, a layer for the application software, a layer for network services, and finally a layer dealing with the hardware interfaces required by the system. Each layer is isolated from the other layers and communicates by means of messages. A layer has no knowledge of the internal operations of another layer and is "a world unto itself" (Rule, 1993). Each layer is, or can be, made up of a set of peer subsystems that are themselves loosely coupled. In this view of a system, each layer serves the layer above it and acts as the client to the layer below it, thus, each subsystem provides some service to the clients of the layer, this is illustrated in Figure. 4.2. which is reproduced from the SIG proposal. The SIG proposal contends that the specification of a system can be viewed as being composed of multiple *realms*. Each layer is a realm, peer subsystems can be viewed as realms only if there is no coupling between them other than messages. The SIG proposal asserts that as each realm is constructed so as to be separate from each other realm in the system, merely communicating via events and messages, then each realm must be treated as a separate system in its own right. Thus each realm is seen as consisting of its own processes and data, these being the logical, dynamic and static views of the realm.

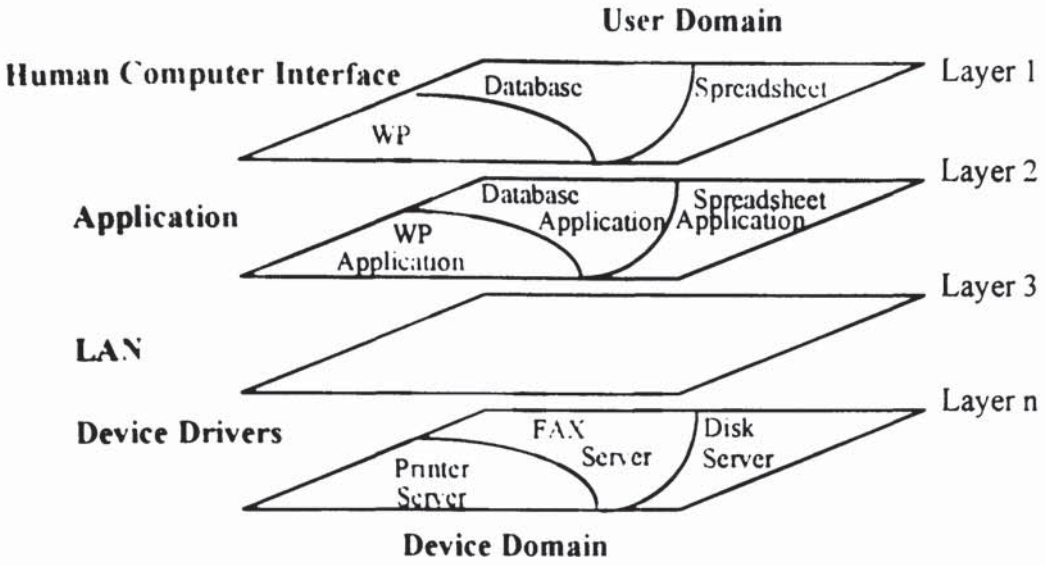


Figure 4.2 The services provided by layers and peer subsystems

Starting with this preliminary exploration of a system the SIG then propose a method that essentially amounts to applying MKII Function Points to *each* realm of the specification The exception to true MKII FPA is that instead of counting the inputs to the realm they propose counting the inputs to each process in each realm The method then consists of several stages:

- For each process type count the number of input message types (IMT) and for each IMT count the input attribute types (IAT) thus for each realm the information processing size due to input messages is given by:

$$IT_r = \sum_{1stProcess}^{nthProcess} \left(IMT + \sum_{1stIMT}^{nthIMT} IAT \right)$$

- For each realm the information processing size due to messages that exit the realm is given by: for each process type count the output message types (OMT) that transit the realm boundary and count each output attribute type (OAT) for each identified message type.

$$OT_r = \sum_{1stProcess}^{NthProcess} \left(OMT + \sum_{1stOMT}^{nthOMT} OAT \right)$$

- To calculate the information processing size of a realm due to all the entity references (ER) made within the realm, count each entity reference made by each of the identified processes thus:

$$ER_r = \sum_{1st\ldots ess}^{nthProcess} ER_p$$

- Calculate the information processing size of a realm as:

$$S_r = IT_r + ER_r + OT_r$$

- Finally count the information processing size of a system as:

$$S_s = \sum_{1stRealm}^{nthRealm} S_r$$

The technique has one departure from simply applying MKII FPA to each realm. This departure is that when counting input types to a realm, all input message types to a process are counted even if they are from other processes in the realm. To justify this, appeal is made to the McCabe (1976) cyclomatic complexity metric and several authors on structured programming (Dhal et, al 1972; Jackson 1975). The SIG asserts that the complexity of a process is a

function of the number of message types it receives, as this determines the number of branches in the program, and the number of data attributes, as this contributes to in-line code operations required to update the data state of the process.

Comments on Function Points for Highly Constrained Systems

The proposal by the UFPUG Extensions SIG is claimed to have "proved useful in a few cases" UFPUG (1993). However no supporting evidence is included in the draft proposal, and the SIG have appealed for developers of real-time systems to provide them with access to data so that the proposed metric can be validated. Despite this the metric would certainly provide a measure of system size, there are however some criticisms which can be levelled at it. The main one of these is that the counting requires a considerable amount of detailed design work to have been undertaken before it can be counted. It is also arguable that the number of connections between processes in a realm is a function of the design choices made rather than the functionality requested by the user i.e., the modularity of a system is determined by the designer in accordance with good design practice not by considerations of functionality.

In one published attempt to use the proposed counting rules to size a system (Sheppard and Turner, 1993) the authors report that their results show that "there does not appear to be a reliable relationship between the EFPUG real-time (sic) FPs and actual effort for the projects under investigation." The authors assert that expert judgement gave a better estimating performance.

4.7 A Proposed Method for Top-Down Sizing of JSD Specifications

The methods that have been examined in this chapter have all attempted to take alternative dimensions of the system specification into account in attempting to arrive at an estimating or sizing model for real-time or scientific software systems. In attempting to derive an estimating model for systems developed in a JSD environment, it will be useful to start by examining JSD specifications, to see how they embody DeMarco's three dimensions of system specification:

- During the entity modelling stage, JSD is concerned with the events that reflect state changes in the problem domain; each event is abstracted within the model as an action. These actions are used along with a knowledge of the problem domain to construct an entity life history model of the real-world entity, thus modelling the state change sequence of the real-world entity that are relevant to the system being specified. The complete JSD entity model is therefore a model of the states and transitions of the problem domain, or that portion of it of relevance to the system being specified. The JSD entity model therefore represents a specification in the control space dimension of DeMarco's three perspectives.

- Each action entering the system boundary is accompanied by its attributes. These attributes are the data generated by the real world event and are the necessary information from the problem domain that the system must process. The total set of attributes of each action is that from which the state vectors of the model processes is synthesised. The state vectors of

these model processes therefore represent the permanently retained data model of DeMarco's perspectives. However the retained data does not represent all the data in the problem space, in order to account for all data it is necessary to count the data attributes associated with all inputs and outputs. This necessity to count of all the data in the system is acknowledged by DeMarco in his Bang metric model when he proposes that the data input elements — "those moving from the manual (data) primitives to the automated primitives" — and the data output elements — transiting in the opposite direction (DeMarco, 1982).

- Functions are added to a JSD specification in the network stage when the functionality required by the user is specified as a network of communicating asynchronous sequential processes, interacting with the entity model processes derived during the modelling stage. This network which captures the functions of the system and their interconnections represents a specification of the function space of the system and corresponds to DeMarco's third dimension (the function space) in his perspective model of system size.

In order to estimate system size using the three dimensional DeMarco model of a specification it is necessary to devise a metric to capture the size of each of the dimensions. The three metrics proposed are: the Control Space Metric (CM) the Information Space Metric (IM) and the Function Space Metric (FM). It will now be useful to examine a JSD Specification to determine which metrics can

be used to adequately capture the size of each of these three dimensions. The system to be examined is a small portion of a specification for a RADAR simulator (Rollo, 1991).

- The JSD specification captures control space dimension as an entity life history model of the real-world entity that is of relevance to the system. In the case of the RADAR simulator, the diagrammatic representation of the entity in Figure 4.3 is in the form of an entity life history diagram, which can, trivially, be represented as a state transition diagram this allows us to count directly the states and transitions of relevance to the problem domain. It is therefore proposed that the control space metric be a simple count of the states and transitions represented by the entity life history models of all the entities and entity roles in the system. Note that this applies even where entities share common actions or they are marsupial entities. Therefore count states and transitions (ST, TR).

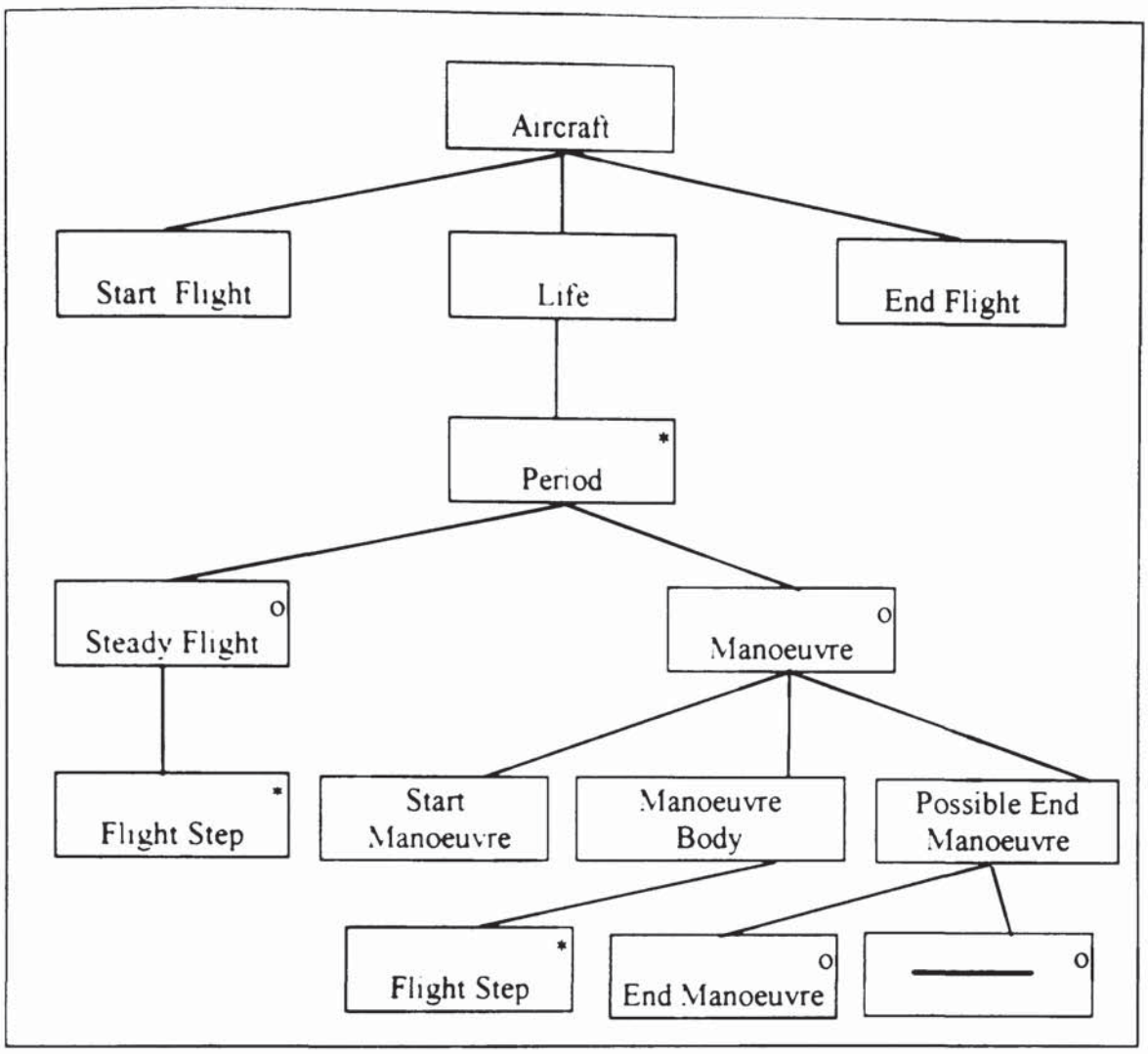


Figure 4.3. An Entity Life-History from a JSD Specification

- An appropriate metric for the information space dimension is, as has been previously discussed, available to us by simply counting the number of data attribute types associated with each input and output that crosses the system boundary (IT, OT).

- For the final dimension of the DeMarco model of a system, we need to size the function space of the system. The earlier discussion relating JSD to this perspective reveals that the network and its connections will provide an insight into the control space. Taking the network illustrated

in Figure 4.4, we see that network connections are characterised as being of two types:

◊ State vector connections;

◊ Data stream connections

Therefore we can represent the control space dimension by a simple count of the number of data stream and the number of state vector connections. It should be noted that the network illustrated is a simple one and does not utilise the full range of connection types available in JSD. There are two variations to the simple data stream connection these are:

◊ Controlled data streams.

◊ Conversational data streams.

Both of these data stream types allow for communication in two directions. A controlled data stream is, effectively a combination of a data stream and a state vector connection, while a conversational data stream permits two-way exchanges of data in which the records are strictly alternate in direction (LBMS, 1991). As each of these effectively represents twice the connectivity of a single data stream or state vector, they should be counted so as to reflect this. Therefore, count a conversational data stream as two data streams and a controlled data stream is counted as one data stream and one state vector interconnection.

There is an additional complication that needs to be borne in mind, namely that two or more data streams entering a process may be combined in two different ways. These are:

- By a fixed merge in which there is a fixed sequence of reading the data stream records;
- By a rough merge in which a single read operation reads input from both data streams in whatever order the records arrive.

While these data stream connections are treated differently by the programmer, they simply represent ways of dealing with multiple data streams entering a process, and for our purposes do not affect the count of data streams.

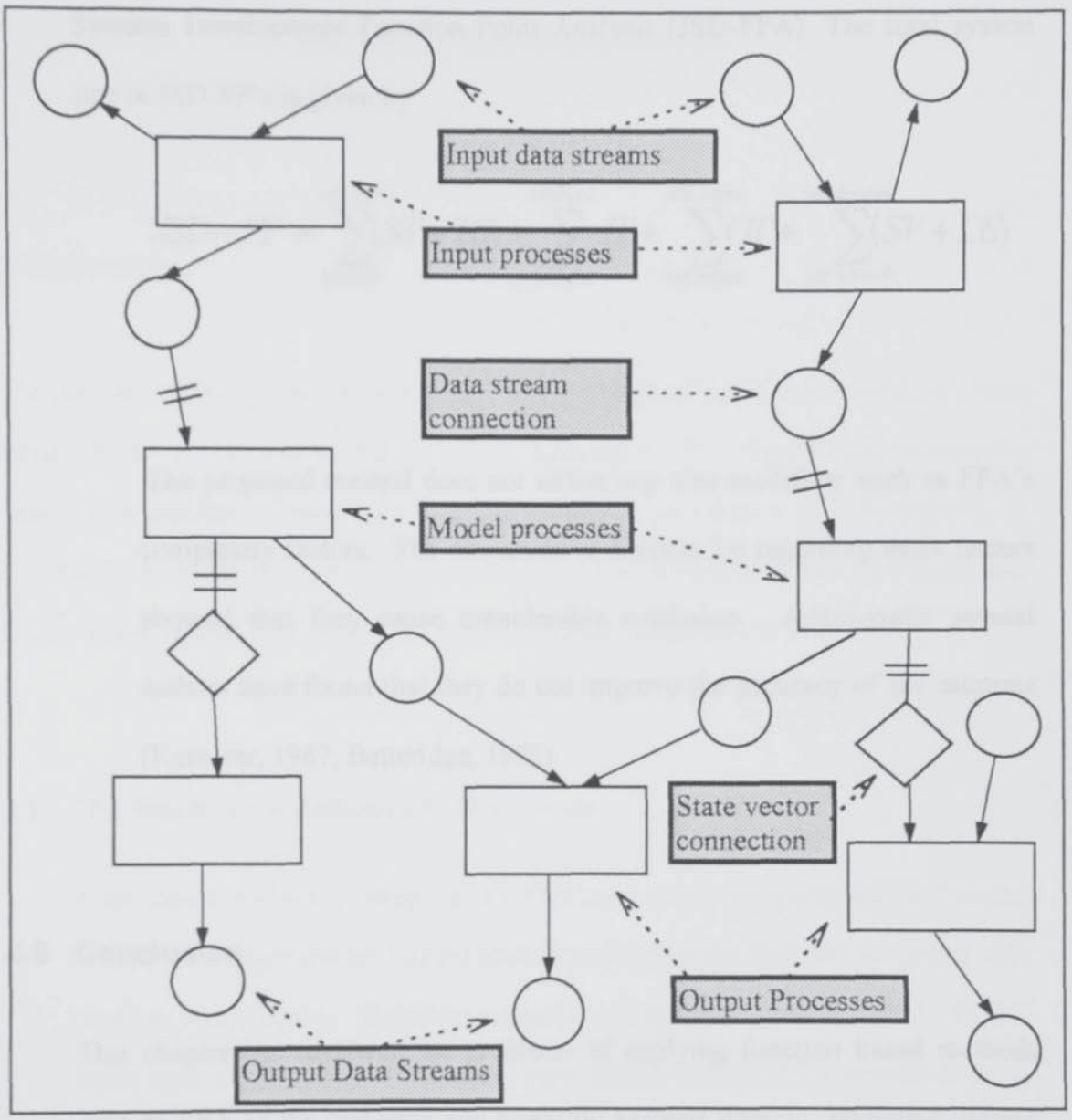


Figure 4.4 The Architecture of a JSD Specification

The control space metric consists of two elements; we count, according to previous rules, all data stream (DS) and all state vectors (SV).

The metric made up from by summing the counts of the metrics (ST, TR, IT, OT, SV and DS) is called a JSD-FP, and the technique is called Jackson Systems Development Function Point Analysis (JSD-FPA). The total system size in JSD-FP's is given by

$$JSD-FP = \sum_{1st\,Flow}^{nth\,Flow}(ST+TR) + \sum_{1st\,Input}^{nth\,Input} IT + \sum_{1st\,Output}^{nth\,Output} OT + \sum_{1st\,Network}^{nth\,Network}(SV+DS)$$

The proposed method does not utilise any size modifiers such as FPA's complexity factors. The discussion in Section 2.6 regarding these factors showed that they cause considerable confusion. Additionally several authors have found that they do not improve the accuracy of the estimate (Kemerer, 1987, Betteridge, 1992).

4.8 Conclusion

This chapter has reviewed the problems of applying function based methods such as FPA to the real-time and scientific systems domain, reviewed various attempts to improve FPA and briefly commented on the efficacy of the latter. Finally, a new method of sizing a JSD specification JSD-FPA, based on DeMarco's three dimensions, has been proposed. The size metric produced by the proposed technique needs to be capable of being turned into an estimate of effort and this will be discussed in Chapter 6.

Chapter 5

Bottom-Up Estimating For JSD

Introduction

The purpose of this chapter is to discuss bottom-up estimation in the context of a JSD environment, and to propose a suitable new technique. The chapter first reviews the need for a bottom-up estimating method and then provides a full description of detailed COCOMO, a method of sizing based on JSD attributes is then discussed, and finally a new model based on COCOMO using JSD parameters is presented.

5.1 The Need for a Bottom-Up Technique

A top-down method to estimate in a JSD context has been outlined in Chapter 4. Many protagonists of cost estimation methods claim that the technique *they* propose is sufficient. However, several authors (e.g. Symons, 1991; CCTA 1991) have asserted that the cost estimator should utilise both a top-down and a bottom-up method. This argument is based on the premise that a top-down method gives the more accurate estimate of project effort early in the life-cycle, whereas a bottom-up method gives a more accurate estimate of the effort associated with individual tasks or system components. Consequently, as a project progresses through its life-cycle, the refinement of previously calculated bottom-up estimates should give a more accurate picture of total effort than initially obtained by whatever top-down method was employed (CCTA, 1991). Also, this refinement provides a useful check on project progress since significant deviations from initial estimates may be detected in advance.

Combining top-down and bottom-up estimation methods has other advantages. For example, it can improve confidence in the estimates obtained, given that the two methods are in tolerable agreement with each other. Combining methods is also useful in hybrid applications where a large element of real-time processing is contained within a business application. As indicated in Chapter 4 experience suggests that a top-down method such as FPA is at its most effective in the business systems applications area, but may not give reliable estimates if, say, a significant real-time element is present.

5.2 Detailed COCOMO

In his development of COCOMO, Boehm (1981,1984) proposed three models. These have been discussed briefly in Chapter 3. It is now appropriate to review the detailed model in some depth as a suitable bottom-up method for JSD. The detailed model becomes useful when the system design has advanced to the point where a complete module-level description of the system exists. In a JSD environment this point is when the system specification has been completed so that all the asynchronous processes in the network have been described.

The COCOMO detailed model provides a method to prepare fine grained estimates which can be processed efficiently. This is done by employing a three-level hierarchical decomposition of the software system to be estimated. The cost drivers whose variation affects the lowest (module) level are considered for each module; these are: complexity, the programmers' capability and experience with the language, and the virtual machine on which the software is to be constructed. At the second level (subsystem), those cost drivers which vary from subsystem to subsystem, but are the same for all modules in the subsystem are considered. These include such things as time and storage constraints, the capability and experience of the analysts, the

reliability and data base size. At the system level, major overall project factors such as nominal effort and schedule equations are applied. The module and subsystem level cost drivers are given in table 5.1.

Table 5.1 Detailed COCOMO cost drivers

Level	Cost Driver	Description
Module	CPLX	Module complexity
	PCAP	Programmers' capability
	VEXP	Virtual machine experience
	LEXP	Language experience
Subsystem	RELY	Required reliability
	DATA	Data base size
	TIME	Computer turnaround time
	STOR	Main storage constraint
	VIRT	Virtual machine volatility
	TURN	Computer turnaround time
	ACAP	Analyst capability
	AEXP	Analyst experience
	MODP	Modern programming practices
	TOOL	Use of software tools
	SCED	Required development schedule

The other feature of the detailed COCOMO model is the use of phase-sensitive effort multipliers to reflect accurately the effect of cost drivers on the various phases of software development. For example, a low level of applications experience will mean additional effort at the early stages of a project. By the

later stages, the team will have built up experience and will require less background learning activity or generate false starts. On the other hand, a slow computer turnaround time will have little effect on the early activities of the project, but will consume more time at the code and testing stages of the project. Detailed COCOMO provides a series of tables, which allocates each cost driver a separate effort multiplier rating to account for its effect on each of the major development phases.

The full procedure consists of 25 steps which are related to the forms which Boehm has designed to record the detail calculations. A summary of the procedure adopted in performing a detailed mode COCOMO estimate is as follows:

- For each module:
  - ◊ Estimate the expected delivered source instructions (EDSI);
  - ◊ Utilising the phase-sensitive effort multiplier table for each cost driver, allocate a rating to each of the module-level drivers, for each phase of development;
  - ◊ Calculate the overall effect of the cost drivers for each phase (EAF M);
  - ◊ Calculate the nominal effort (MM NOM) by using the appropriate COCOMO formula and the phase distribution of effort from the appropriate tables (an example is Table 3.1);
  - ◊ Using MM NOM and EAF M, calculate the estimated effort for each phase of development (MM MOD);
- For each subsystem:

- ◊ Transfer from the module level estimates:
  - * the total EDSI
  - * the total MM MOD;
- ◊ For each subsystem level cost driver estimate the effort multiplier value for each phase of development, using the tables provided;
- ◊ From the individual cost driver values calculate the overall effort multiplier value by development phase (EAF SS);
- ◊ From total MM MOD and EAF SS calculate the estimated effort (MM EST);
- ◊ Calculate the money cost.

- For the system as a whole:
  - ◊ Sum the MM EST values to obtain a total effort estimate for the project (MM);
  - ◊ From the MM value and the appropriate formula calculate the nominal schedule for the project;
  - ◊ Sum the money cost for each subsystem to obtain the money cost for the whole project.

This completes the estimate, though it is possible for the project manager to go further and calculate the required schedule for each module. Additionally, as staff are allocated to each module, it is possible to make revisions to the estimates if the personnel cost drivers (PCAP, LEXP, etc.) vary.

5.3 Sizing Modules Using JSD Attributes

The sizing models developed in this chapter are based on a set of metrics available at a relatively early life-cycle stage in the JSD method (Cameron, 1988; Jackson, 1983). The metrics are derived from the process descriptions of a JSD specification. However, although the models are tailored to JSD, the possibility exists that they could be extended to other methods (such as SSADM Version 4 (NCC, 1990)) which make use of similar process description techniques.

JSD process structures are pure tree hierarchies typically represented in diagrammatic form and comprise three basic node types: sequences, selections and iterations as illustrated in Figure 5.1, (one can categorise JSD's specialised posit-admit 'backtracking' structures as selections). The proposed metrics are based on the 'boxes' representing the nodes of structure diagrams. Two obvious simple metrics are:

- The total number of boxes (BX)
- The number of levels (LV).

However, it is possible that less coarse-grained, but still relatively simple, metrics might prove to be more effective as size predictors; for example:

- The totals of leaf (LF) and non-leaf (NL) nodes
- The totals of node types, i.e., sequences (SQ), selections (SL), and iterations (IT).

The last three metrics seem attractive since, unlike the other metrics which treat process structures as simple graphs, node type totals are in some sense

indicators of the 'structural complexity' of a process. In practice the control elements of the final code can be mapped directly to the selection and iteration nodes of the process structure diagram.

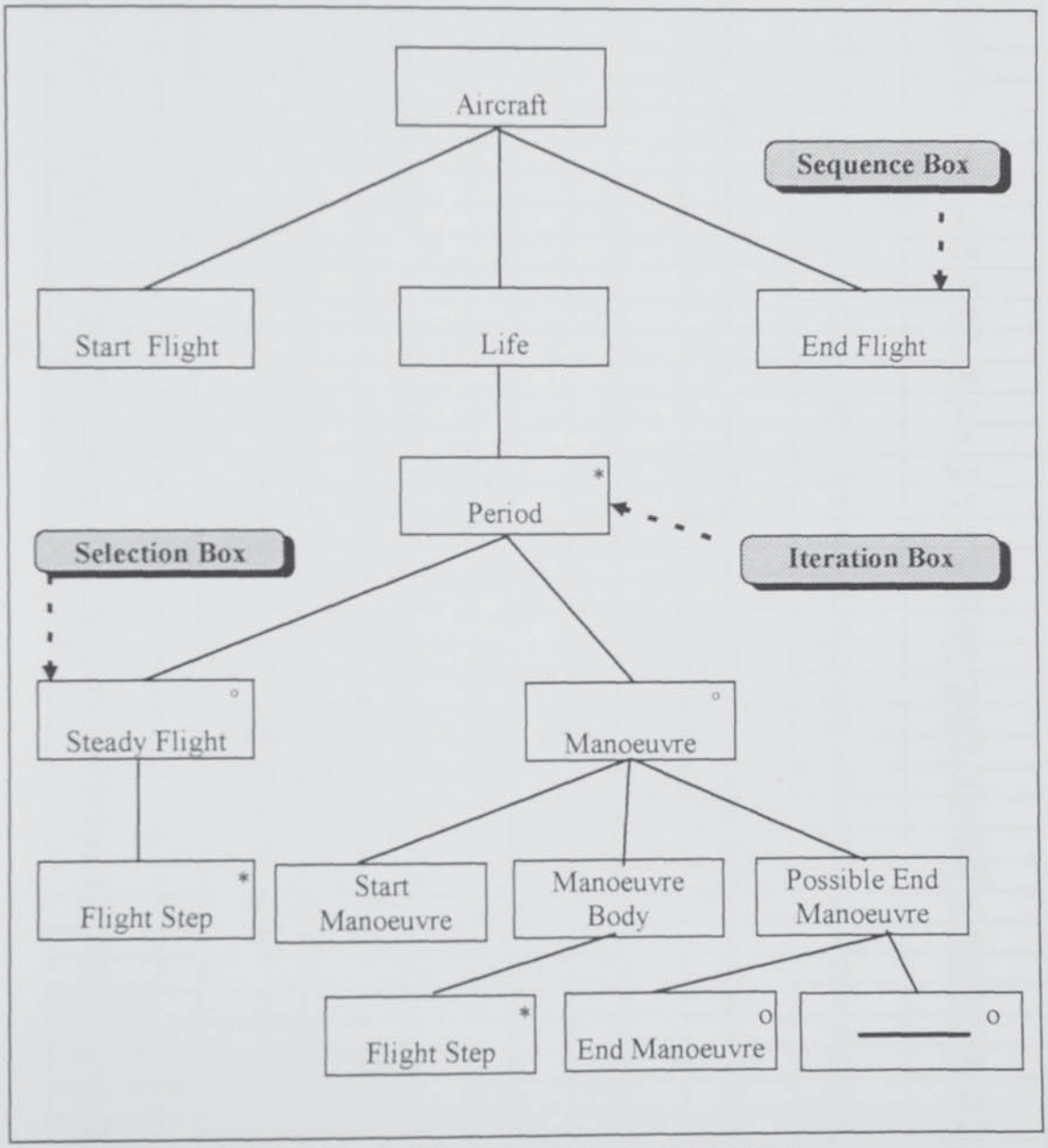


Figure 5.1 A JSD Process Structure Diagram

Table 5.2 Original Data Set used to derive models								
Cases	SL	IT	SQ	LV	LF	NL	BX	DSI
1	2	1	7	7	8	7	15	115
2	5	1	19	7	21	20	41	186
3	0	0	6	3	3	3	6	31
4	1	0	7	4	5	6	11	22
5	2	0	4	5	5	5	10	153
6	2	1	7	7	7	10	17	60
7	1	0	25	4	25	23	48	137
8	1	0	12	2	6	10	16	43
9	0	0	6	3	3	4	7	26
10	3	1	11	6	7	9	16	178
11	2	1	10	7	8	7	15	301
12	1	1	7	4	6	7	13	108
13	0	0	5	2	5	1	6	876
14	2	1	11	7	11	6	17	250
15	2	1	12	7	11	6	17	317
16	0	0	6	2	5	1	6	136
17	2	4	12	9	31	9	40	705
18	0	0	7	3	4	3	7	88
19	3	5	34	11	24	28	52	682
20	3	3	42	9	28	26	54	807
21	0	0	7	3	4	3	7	94
22	1	0	6	4	3	5	8	257
23	1	0	9	4	4	6	10	324
24	1	0	8	4	6	8	14	456
25	3	0	9	5	9	13	22	678
26	3	0	9	5	9	13	22	661
27	3	0	9	5	9	13	22	666
28	1	0	3	4	3	5	8	274
29	1	0	4	4	4	6	10	344
30	1	0	4	4	4	6	10	314
31	2	0	7	5	7	10	17	518
32	3	0	14	5	14	18	32	1028
33	1	0	5	4	5	7	12	392
34	1	0	3	4	3	5	8	264
35	3	0	11	5	10	15	25	830
36	1	0	3	4	3	5	8	301
37	1	0	3	4	3	5	8	272
38	5	1	17	8	23	19	42	689
39	1	4	31	14	26	26	52	362
40	9	16	37	9	57	39	96	563
41	4	1	16	6	11	13	24	216
42	21	4	59	13	55	61	116	644
43	1	2	10	6	8	13	21	131
44	2	1	12	5	11	15	26	298
45	1	3	23	11	16	19	35	488
46	0	1	3	7	8	5	13	221
47	3	5	15	10	19	24	43	654
48	0	1	6	7	17	10	27	188
49	2	1	4	7	8	6	14	426
50	1	2	17	15	18	20	38	524
51	10	2	17	9	29	19	48	520

As a preliminary manoeuvre, it was decided to establish the suitability of the various metrics as DSI sizing predictors. This was carried out by reference to a data set pertaining to some 51 processes in total, each of which was associated with a single financial business system. This enabled all of the above metrics to be collected, together with the actual DSI counts of the corresponding program modules derived from the process structures. The DSI counting rules used were those described by Boehm (1981).

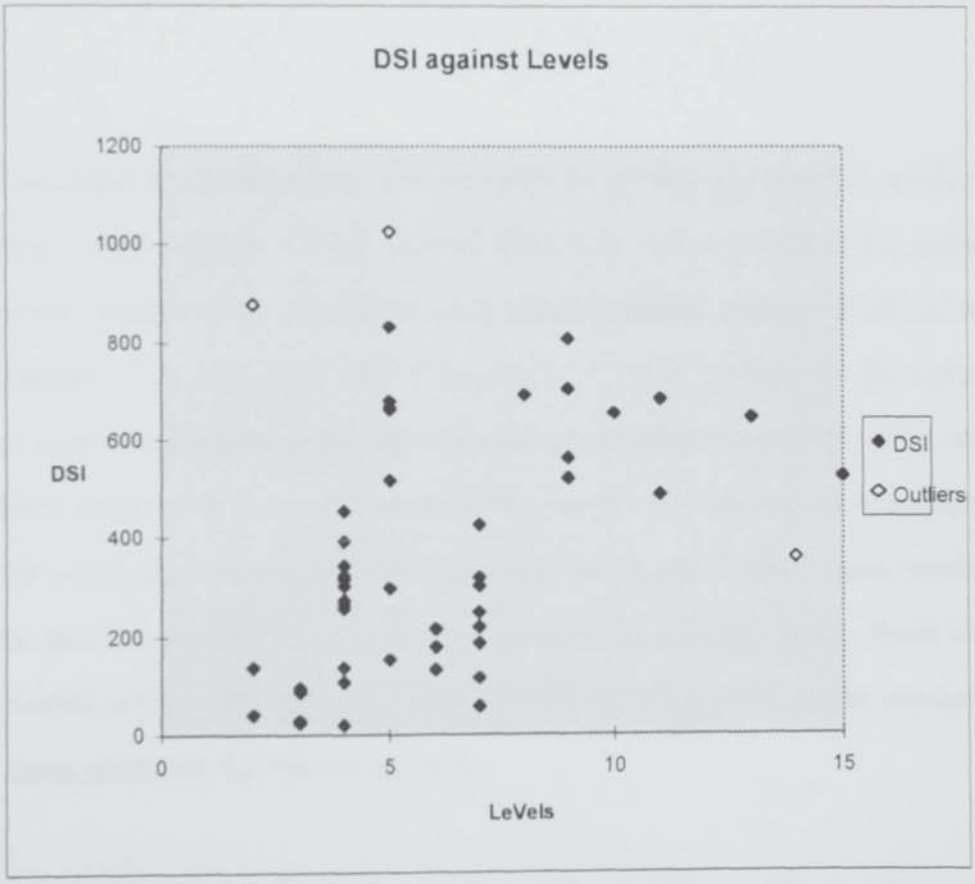


Figure 5.2 Scatter Plot for DSI against LV

The data set was analysed in various ways. Firstly a simple scatter plot of each metric against DSI was produced, the full set of scatter plots are given in Appendix E (Figure 5.2 shows the scatter plot for DSI against LV). This was

followed by the correlation of DSI with each of the candidate metrics producing the following coefficients:

	BX	LF	NL	SQ	SL	IT	LV
DSI	0.786	0.764	0.79	0.762	0.508	0.762	0.832

As can be seen, the highest degree of correlation was obtained between LV and DSI, although the data is well scattered, and LV could be regarded as a 'low resolution' metric which might be too coarse to be an effective predictor on its own.

An examination of the scatter plots revealed the presence of a small number of outliers. Investigation of the outliers identified one case where the process structure pertained to a program element which consisted almost entirely of data declarations describing the file structures to be used throughout the system. This case was atypical of the data set and was therefore removed. The other outliers were not so easily explainable, but simple box plots and an examination of the residuals of linear regression revealed that the three other cases involved were also extreme (in excess of three standard deviations); hence, these were also removed from the data set. Recalculations of correlation on the remaining 47 cases produced the following results:

	BX	LF	NL	SQ	IT	SL	LV
DSI	0.859	0.827	0.864	0.797	0.627	0.783	0.841

Removal of the outliers caused the correlation between DSI against BX, LF and NL to increase and match that of LV, the full set of statistics are include in Appendix C

5.4 Sizing Models

The initial assumption made was that DSI would be best predicted by a model based on the number of sequences, selections and iterations making up a structure diagram. The reason for taking this view was that these constructs have an obvious direct mapping to the procedural language code, with the associated metrics (SQ, IT and SL) in effect treating a structure diagram as something more than just a simple graph. However, because high correlation's were observed between DSI and such metrics as BX and LV, it was decided that models made up of the full range of metrics should be explored.

Altogether, seven DSI sizing models were constructed. These models were investigated using linear regression on the data set, there being no evidence of a non-linear relationship from inspection of the scatter plots of the various metrics. The first four of these models, see Table 5.2, represented what were considered to be the most straightforwardly obvious contenders. Each model involves one or more 'logically related variables' (rather than unrelated variables such as NL and IT), and the variable(s) in each model in some sense 'covers' the whole of a process structure. The results of the linear regression are included in Table 5.2.

Table 5.3 Original Potential DSI Sizing Models

Model Identifier	Model
M1	$DSI = a \times SQ + b \times SL + c \times IT$ $a = 14.4, b = 72.2, c = 5.4$
M2	$DSI = a \times BX$ $a = 14$
M3	$DSI = a \times LF + b \times NL$ $a = 4, b = 24.4$
M4	$DSI = a \times LV$ $a = 54.5$

However, as the variable LV displayed strong correlation with DSI, it was felt that models based on M1– M3 which include LV as an additional factor might prove to be more accurate. LV is in some sense a 'different' measure to the other variables, which are concerned directly with box counting. The incorporation of LV will cause the model to capture an extra dimension of process structures, namely their depth. Thus, the alternative models described in Table 5.3 were also investigated, with the results as shown.

Table 5.4 Additional Potential DSI Sizing Models

Model Identifier	Model
M5	$DSI = a \times NL + b \times LV + c \times LF$ $a = 191, b = 188, c = 0.6$
M6	$DSI = a \times BX + b \times LV$ $a = 9.3, b = 20.4$
M7	$DSI = a \times SQ + b \times SL + c \times IT + d \times LV$ $a = 6.3, b = 49.7, c = -35.2, d = 35.3$

Analysis

Full multiple regression on the seven proposed models was carried out. Examination of the results (Appendix C) revealed that incorporating LV as an extra variable into models M1—M3 helped to explain, very little additional variance in the data. The effect on each of the models is given in Table 5.4. In fact, the univariate models M2 and M4 accounted for 73% and 70.% respectively of the variance in the data. Hence the tentative, and somewhat unexpected, inference drawn from this analysis was that a simple sizing model was probably sufficient for DSI estimation.

Table 5.5 Effect of Adding LV to Models M1-M3	
Model	**Additional variance explained**
M1	4.4%
M2	0.83%
M3	0.64%

To test this inference further, a set of DSI estimates was prepared using each of the seven models in turn on the 47 process cases. The estimates were then compared using a paired T-Test. The null hypothesis was that the estimates would produce a data set similar to that actually obtained by generating code from the process descriptions — more specifically, that the DSI figures and the estimates would show no difference at the 0.1 level of significance. At this level, it is more likely that a valid model would be rejected than that an invalid model would be accepted. The results of the T-Test showed that the null hypothesis could not be rejected.

The seven models were then used to generate a further set of estimates, this time using a different data set. This data set concerned 24 processes and was obtained from a separate project, though the system involved was again a financial business system. T-Test comparisons were once again carried out to test the same null hypothesis—that the estimates and the actual DSI figures were from two populations which have the same mean values. The results showed that the null hypothesis could not be rejected in the case of one model only, namely M4. This result substantiated the inference drawn earlier that a

simple univariate model is the 'best bet' for deriving DSI estimates from process structure diagrams.

Table 5.6 Data Set Used to Validate Models

Cases	SL	IT	SQ	LV	LF	NL	BX	DSI
1	148	2	102	8	63	188	251	1332
2	6	0	17	6	14	9	23	152
3	26	0	26	4	6	27	33	136
4	4	0	11	5	8	7	15	139
5	25	1	13	5	10	29	39	149
6	8	1	9	5	6	12	18	134
7	19	2	13	5	10	24	34	143
8	2	2	13	7	8	9	17	140
9	2	1	12	5	8	7	15	138
10	0	1	9	5	6	4	10	135
11	0	0	8	3	6	2	8	137
12	0	2	36	7	13	26	39	189
13	0	2	36	7	26	13	39	197
14	2	0	5	4	6	3	9	135
15	6	1	16	5	14	9	23	154
16	5	1	23	5	21	8	29	173
17	2	2	10	5	8	6	14	142
18	112	1	144	8	108	149	257	798
19	9	1	19	7	14	15	29	315
20	7	0	20	8	14	13	27	305
21	66	0	71	4	67	70	137	403
22	51	0	79	9	63	68	131	573
23	0	0	8	3	6	2	8	137
24	20	1	42	10	32	31	63	413

The result that model M4 provides an acceptable forecasting model is fortunate as the multivariate models are made up of variables that exhibit a strong multicollinearity. This level of multicollinearity can cause the prediction of such a model to be subject to errors (Bowerman & O'Connell, 1990).

A further validation of the models was carried out, the forecast values were compared to the actual figures from data set 2 and the absolute and mean relative errors were generated (Table 5.7, 5.8 and 5.9, 5.10 respectively).

Table 5.7 Absolute Errors in Forecasts

M1Fcast	M2Fcast	M3Fcast	M4Fcast	M5Fcast	M6Fcast	M7Fcast
10833.20	2182.00	3507.20	896.00	2447.00	1165.50	6878.20
526.00	170.00	123.60	175.00	141.10	184.30	465.10
2115.60	326.00	546.80	82.00	458.50	252.50	1461.20
308.20	71.00	63.80	133.50	93.50	102.50	305.60
1848.60	397.00	598.60	123.50	504.90	315.70	1316.70
578.60	118.00	182.80	138.50	192.80	135.40	461.60
1426.80	333.00	482.60	129.50	415.40	275.20	989.30
202.40	98.00	111.60	241.50	168.30	160.90	218.00
184.60	72.00	64.80	134.50	94.50	103.50	178.30
0.00	5.00	13.40	137.50	39.00	60.00	63.00
21.80	25.00	64.20	26.50	38.80	1.40	19.30
340.20	357.00	497.40	192.50	447.00	316.50	214.50
332.20	349.00	224.20	184.50	198.50	308.50	206.50
81.40	9.00	37.80	83.00	1.10	30.30	137.10
515.00	168.00	121.60	118.50	120.30	161.90	386.30
524.60	233.00	106.20	99.50	86.40	198.70	361.70
157.20	54.00	36.40	130.50	71.40	90.20	126.50
9367.40	2800.00	3269.60	362.00	2263.10	1755.30	5922.80
613.80	91.00	107.00	66.50	111.50	97.50	463.90
488.40	73.00	68.20	131.00	102.10	109.30	451.30
5384.60	1515.00	1573.00	185.00	1049.40	952.70	3465.70
4246.80	1261.00	1338.20	82.50	932.80	828.90	2777.10
21.80	25.00	64.20	26.50	38.80	1.40	19.30
1641.20	469.00	471.40	132.00	386.30	376.90	1163.40

Table 5.8 Mean Absolute Error in Forecasts for Each Model

M1Fcast	M2Fcast	M3Fcast	M4Fcast	M5Fcast	M6Fcast	M7Fcast
1740.02	466.71	569.78	167.17	433.44	332.71	1168.85

The absolute and relative error tables demonstrate that model M4 achieves the closest forecast, with an MRE of 0.71. This error reveals that the forecast will contain considerable individual errors but that if treated as a first estimate of DSI for a

system then it will provide a useful first estimate. This estimate should be revised as more data is available from module coding activities.

Table 5.9 Relative Errors in Forecasts

M1Fcast	M2Fcast	M3Fcast	M4Fcast	M5Fcast	M6Fcast	M7Fcast
8.13	1.64	2.63	0.67	1.84	0.88	5.16
3.46	1.12	0.81	1.15	0.93	1.21	3.06
15.56	2.40	4.02	0.60	3.37	1.86	10.74
2.22	0.51	0.46	0.96	0.67	0.74	2.20
12.41	2.66	4.02	0.83	3.39	2.12	8.84
4.32	0.88	1.36	1.03	1.44	1.01	3.44
9.98	2.33	3.37	0.91	2.90	1.92	6.92
1.45	0.70	0.80	1.73	1.20	1.15	1.56
1.34	0.52	0.47	0.97	0.68	0.75	1.29
0.00	0.04	0.10	1.02	0.29	0.44	0.47
0.16	0.18	0.47	0.19	0.28	0.01	0.14
1.80	1.89	2.63	1.02	2.37	1.67	1.13
1.69	1.77	1.14	0.94	1.01	1.57	1.05
0.60	0.07	0.28	0.61	0.01	0.22	1.02
3.34	1.09	0.79	0.77	0.78	1.05	2.51
3.03	1.35	0.61	0.58	0.50	1.15	2.09
1.11	0.38	0.26	0.92	0.50	0.64	0.89
11.74	3.51	4.10	0.45	2.84	2.20	7.42
1.95	0.29	0.34	0.21	0.35	0.31	1.47
1.60	0.24	0.22	0.43	0.33	0.36	1.48
13.36	3.76	3.90	0.46	2.60	2.36	8.60
7.41	2.20	2.34	0.14	1.63	1.45	4.85
0.16	0.18	0.47	0.19	0.28	0.01	0.14
3.97	1.14	1.14	0.32	0.94	0.91	2.82

Table 5.10 Mean Relative Error for each Model

M1Fcast	M2Fcast	M3Fcast	M4Fcast	M5Fcast	M6Fcast	M7Fcast
4.62	1.28	1.53	0.71	1.30	1.08	3.30

5.5 Model M4 as an Input to SLOC Based Estimation

The detailed model COCOMO described earlier requires a detailed breakdown of the system to be estimated. Using model M4 described in the last section provides us with a set of SLOC estimates at the required level of detail. In a JSD specification, it is usual to specify the system as a collection of sub-networks of asynchronous processes, each sub-network being based on some function. These processes coincide with the module-level estimates of the three-level hierarchy that detailed COCOMO requires. It was therefore proposed that the module-level estimates in detailed COCOMO are prepared using the outputs of model M4. The subsystem level of detailed COCOMO is similar to the separate sub-networks that are described in a JSD specification. It was therefore proposed that the subsystem estimates are based on these sub-networks. This would produce a detailed COCOMO model using the outputs of model M4 as its input parameters. If the estimator wishes to use the intermediate COCOMO model that would also be possible using the output of model M4 as a set of inputs.

The detailed effort multipliers applicable to JSD can be obtained from Table 5.1 substituting process for module and sub-network for subsystem. The justification for this substitution is on the statement by Boehm (1981) that the module level cost drivers may vary from one module to another within a subsystem and that a subsystem is generally made up of modules that have some related function, and the general practice of JSD designers to partition the overall network into sub-networks also based on function.

5.6 M4-driven JSD-COCOMO

In order to produce an M4-driven version of detailed COCOMO, it is necessary to produce tables for the phase-sensitive module and sub-network effort multipliers. The multiplier values are derived from the equivalent tables provided by Boehm (1981), using a mapping between the work breakdown phases of COCOMO and the work breakdown stages of JSD taken from the JSD manual (LBMS, 1991). The mapping is based on descriptions of the activities of the COCOMO phases (Boehm, 1981) and the activity in each of the JSD stages (LBMS, ibid). The phase/stage relationship used is as follows; the Product Design (PD) phase of COCOMO equates to the Architecture (AR) and Network Specification (NS) stages of JSD, the Detailed Design (DD) phase of COCOMO equates to the Physical Design (PD) stage of JSD, Code and Unit Test (CUT) relates to the Construction (CO) stage of JSD, and finally the Integration and Test phase (IT) of COCOMO equates to the Installation Stage (IN) of JSD. An example of the details of the effort multiplier adjustments are given in Table 5.5 for one module level cost driver (CPLX), the full tables are given in Appendix D. The figures given are nominal, and should be calibrated in the context of the actual development environment using the method detailed by Boehm (1981). The calibration of COCOMO is described by Boehm as affecting various aspects of the models such as the nominal effort equations as well as consolidating the cost drivers. Boehm advises that cost drivers may be calibrated to allow for the uniformity of development environment that might be found in an organisation, the TOOL cost driver would be appropriate within a JSD environment where the same degree of tool usage pervades the development environment. In order to calibrate the actual values given for each cost driver it will be necessary to use the 'ideal effort multiplier' technique that Boehm used to derive the values in the first place, this will require a considerable amount of project data — the original figures are based on a data set of 63 projects. The best approach would be to gather data, monitor the

accuracy of the forecasts and as experience is gained identify those cost drivers which are not performing well and thus limit any recalculation required.

The steps in implementing M4 based COCOMO are:

- For each process in the system to be estimated:
  - ◊ Use model M4 to prepare a SLOC estimate (M4 SLOC);
  - ◊ Utilising the phase sensitive multiplier table (Appendix D, Table 1) allocate a rating to each of the process level cost drivers, for each phase of development;
  - ◊ Calculate the overall effect of each cost driver for each process for each phase (EAF P);
  - ◊ Calculate the effort (MM NOM) by using the appropriate COCOMO formula and the phase distribution of effort from Table 3 3
  - ◊ Using the calculated MM Nom and EAF P prepare an estimate of the effort required for each phase of development (MM P)

Table 5.11 Example Phase Sensitive Effort Multipliers

	Rating	AR	NS	PD	CO	IN
CPLX	v low	0.70	0.70	0.70	0.70	0.70
	low	0.85	0.85	0.85	0.85	0.85
	nominal	1.0	1.0	1.0	1.0	1.0
	high	1.15	1.15	1.15	1.15	1.15
	v high	1.30	1.30	1.30	1.30	1.30
	extra high	1.65	1.65	1.65	1.65	1.65

- For each Sub-Network in the system
  - ◊ Transfer from the process level estimates:
  - ◊ the total M4 SLOC for each sub-network (DSI SN)
  - ◊ the total MM P for each sub-network (MM SN);
  - ◊ For each sub-network level cost driver estimate the effort multiplier value for each phase of the development, using the table in Appendix D (Table 2),
  - ◊ From the individual cost driver values calculate the overall effort multiplier value by development phase (EAF SN);
  - ◊ From MM SN and EAF SN calculate the estimated effort (MM EST);

    Calculate the money cost.

- For the system as a whole:
  - ◊ Sum the MM EST values to obtain a total effort estimate for the project (MM)
  - ◊ From the MM value and the appropriate (COCOMO) formula calculate the nominal schedule for the project
  - ◊ Sum the money cost for each subsystem to obtain the money cost for the whole project.

This procedure will provide a complete bottom-up estimate for a JSD based project. The project manager can, however, utilise other features of detailed COCOMO and prepare estimates of required schedule for each module or sub-network. It is also advisable continually to revise the estimates in the light of

changes, to planned staff for example, and as more accurate estimates of the SLOC for each of the implementation modules becomes available.

5.7 Conclusion

This chapter has explored the detailed mode of COCOMO and outlined an analysis of possible metrics for bottom-up sizing within JSD. A suitable model for sizing a JSD project in terms of process structure diagrams has been proposed. Finally, a method of using the sizing model as an input to COCOMO has been described; additionally, alterations to the COCOMO phase-sensitive cost driver effort multipliers are suggested to reflect the different JSD life-cycle work breakdown structure.

Chapter 6

Applying and Validating the Various FPA-based Cost Models

Introduction

This chapter explores the proposed FPA-based methods of sizing a JSD system, and attempts to justify and validate the proposed metrics. The chapter starts with an examination of a selection of 'real' projects, first utilising the counting rules proposed in Section 3.3, and then the top-down method of Chapter 4. As the number of actual projects available is quite small, the chapter then moves on to explore the method by simulation. Finally, the chapter derives a method by which the sizing metrics may be turned into estimates of effort and cost. The complete FPA-based estimating method thus established is then validated against the 'real' projects that are available.

6.1 Applying the A&G-based Counting Rules to a Commercial MIS

The Albrecht and Gaffney version of FPA, with the mapping suggested in Section 3.3 (Rollo & Ratcliff, 1988), was used within Michael Jackson Systems (UK) Ltd (MJUK) by their consultants to estimate both internal projects and client projects. The method was used to estimate several internal projects, and then two commercial MIS systems, and three phases of a real-time system. However, one of the commercial systems, for an external client, was not subsequently proceeded with by MJUK, so there was no indication of the effectiveness of the estimating method in this case.

The experience with the other commercial system, also developed for an external client, was encouraging, although several points emerged. The

method was originally tried in conjunction with the formulas provided for SLOC and effort by Albrecht & Gaffney (1984). Based on the MJUK experience of using the rules (Rollo & Ratcliff, 1988) to estimate for several internal projects, various modifications were proposed (MJUK, 1989b), as follows:

- Included in the MJUK FPA calculations is an *operations* factor. This factor is introduced to account for the perceived productivity benefits gained from using JSD. Specifically, the benefits were seen to be gained by the method's ability to reuse operations lists of program actions, as these are only written once but typically used several times in development. The value of the operations factor is given as 0.4.

- The formulas provided by Albrecht & Gaffney (**ibid**) do not take account of productivity gains realised by tool support. The experience of MJUK on several client systems was that the use of tools provided productivity gains by a factor of between 2.0 and 2.5.

- A *method* factor was also introduced. This factor takes account of the impact of using a systems development method. In some respects, the use of a method increases the workload due to such things as the inherent formalism. The method factor multiplies the original estimate by 1.1.

Despite the introduction of these factors to adjust the cost estimate, the experience of MJUK on this commercial system (MJUK, 1990) was that there was a considerable overestimate of both the SLOC and the effort. These over-estimates were in the order of a factor of 2 in the case of the SLOC estimate and 1.5 as regards the actual effort expended. MJUK undertook a review of the estimates (included as an addendum to MJUK, 1990) and noted that the original figure for the effort overestimate of 1.5 times only took account of the effort expended by the MJUK personnel; if the effort expended by the external client

was also taken into account the overestimate reduced to a factor of 1.2. As an estimator method should be sure exactly what effort is being estimated this author considers this to be an invalid post-hoc justification.. The review also reports that an early estimate produced before the specification was complete had produced an effort estimate which was some 24% below the actual effort expended. It was also noted that the Albrecht & Gaffney formula for SLOC included comment lines. The output of the program generator being used did not include comments and so an overestimate was to be expected; "SLOC is a suspect term ..as it is difficult to define what a 'source line' is." (MJUK, 1990). The review suggested that an alternative figure for SLOC calculations should be used, this being the figure of 105 COBOL SLOC per function point which was derived from a list of language coefficients published by the International Function Point User Group (IFPUG). The review also suggested that a local productivity figure in terms of function points per hour be produced from this project's data, and that this be used to produce overall effort estimates for future projects.

The conclusions of this (MJUK, 1990) were that:

- Despite the dangers of generalising from a small sample, the FPA technique is at least a viable candidate for early scoping estimates. The early estimate produced was somewhat below the actual effort used, but the technique had produced useful results which were a "significant improvement on the 'guestimates' commonly used."

- Using the language coefficient from the IFPUG figures improved the SLOC estimate, at least for projects in the region of 700-800 function points.

- For projects with similar characteristics it would be useful to compare the effort and SLOC estimates when a 'productivity' based figure was used to generate effort and a language coefficient to produce the SLOC estimates.

Comments

This investigation carried out by MJUK was an extremely useful attempt to use the proposed counting rules in industry to carry out an actual estimate. The estimate was used to guide the development as it highlighted a potential overspend of resources; and as a consequence the work was rescheduled and various minor functions were prioritised into minor increments — though in the event; certain minor increments were delivered late.

In fact, the estimate of FP produced was slightly different to the estimate produced independently by this author utilising the same data. However, the difference was not so severe as to invalidate the company's conclusions from the experience. As a result of this trial and the subsequent publication of the MKII FP method by Symons (1991), it was decided that the idea of basing effort estimates on historical productivity figures might produce more useful results.

6.3 Applying the A&G-based Counting Rules to a Real-Time System

MJUK also attempted to apply the Ratcliff and Rollo (1988) method in the context of real-time systems, though it was expected that this would be problematic. The system chosen was a simulation of a flight control system; the system was to be developed in three increments, each increment being quite small (for example, increment one was 3800 lines of Ada code). Using the mapping suggested by the Ratcliff and Rollo paper this first increment was estimated as being 68 Function Points. The Albrecht & Gaffney formula was used having been adapted to use the IFPUG language coefficient for Ada (71 Ada SLOC per FP). However the result of the effort calculations was to

produce an estimate of work hours which was negative (Rule, 1989)! This result was obtained despite the fact that the estimated SLOC was 4830 as opposed to the actual developed SLOC of 3800. Clearly, therefore, although the estimate of SLOC was within some 30% of the actual, the estimate of work hours was unusable.

This result reinforced the view that figures based on productivity in terms of Function Points would be more useful in the production of work hours estimates. The figures from increment one were therefore used to derive productivity figures which were applied to increment two; the productivity figures overestimated the effort for increment two by a significant amount. The reason for the overestimate of increment two is largely accounted for by the fact that the tools being used were newly developed code generation tools which were designed to transform the JSD specification into executable Ada code. In addition, time effort was expended producing components reusable in subsequent increments. The figures for increment two were then used to produce productivity figures for increment three. The use of these figures also overestimated the effort for the third increment. Again an increase in the productivity of the development team between increments two and three was thought to be the cause. In order to try to verify the suspicion that an increase in the team's productivity was causing the underestimates a comparison of productivity based on the actual SLOC was produced. Using the actual amount of delivered SLOC, it was found that for increment one productivity was 9.2 SLOC per work hour, the average figure for subsequent increments being 22.7 SLOC per work hour. The ratio of the increase in SLOC productivity was similar to that based on the estimated function points namely from 0.16 FP/Hr to an average 0.5 FP/Hr for the subsequent increments (Rule, 1991).

Comments

The MJUK analyst also attempted to produce FP figures using the MKII FPA method of Symons (1988) This produced FP counts which were considerably lower than those produced by the Rollo and Ratcliff mapping of the Albrecht & Gaffney method. However, these counts were considered to be more consistent compared with the actual work hours expended on each increment. The high variation in productivity was investigated by this author and found to be largely due to the reasons given. However, an additional complication was that the software engineer developing the Simulator was also responsible for developing the tools being used, and there is little doubt that some of the variation in productivity can be accounted for by tool 'tuning' activities being undertaken during the first increment being costed to the Simulator project, this being the first attempt to use the tool on an actual project.

6.3 Sizing Projects Using the JSD-FPA Metric

The available projects included three commercial financial systems and two real-time systems, one of which was developed in two increments thus providing in total six separate developments on which to attempt to validate the method.

Pensions

This project was a commercial information system called Pensions. The project was developed by Abbey National Ltd to support the launch of their pension products in 1988. The developers had previously used JSD on two other projects and so software personnel were already familiar with the method. The project team made full use of the tool support available from MJUK, namely Speedbuilder, Program Development Facility (PDF) and JSP-COBOL. In order

to meet the launch date set by the UK Government for the introduction of new pension schemes by building societies, the system had to be developed quickly once the final form of the legislation became clear.

Table 6.1 **Project Name: Pensions**	
States (S)	200
Transitions (T)	215
Input Types (IT)	213
Output Types (OT)	770
State Vectors (SV)	110
Data Streams (DS)	224
Total JSD-FPs	**1732**

Apart from the modelling phase, it took some five months to complete the specification and implementation of the system, along with one major amendment to the JSD model caused by revisions to the proposed legislation before it became law. Data was collected by visiting the company's site, where appropriate personnel were interviewed and various technical documentation analysed. The JSD-FP counts for this system are presented in Table 6.1.

Cheques

This project was also undertaken by Abbey National Ltd, who were developing their banking facilities. This project was started before the Pensions project and was the second JSD project undertaken by the development team. The project was supported by the same range of tools as the Pensions project, though it operated to less constrained time-scales. The data was collected by visiting the company's site and interviewing personnel and examining project documentation. The JSD-FP counts obtained from this project are given in Table 6.2 below.

Table 6.2 **Project Name: Cheques**	
States (S)	254
Transitions (T)	267
Input Types (IT)	877
Output Types (OT)	1019
State Vectors (SV)	559
Data Streams (DS)	572
Total JSD-FPs	**3548**

Sharelink

The Sharelink project was developed by MJUK themselves for an external client who wished to develop an information system for the launch of new companies onto the Stock Exchange. The first company to be launched using this software was Abbey National Ltd. The system was developed to quite tight time-scales. The development team, being personnel drawn from the staff

of MJUK, were all experienced JSD staff. The development team made full use of the tools available to them, which amounted to the same tools as for the previous two projects. Data collection for this project followed the style of the previous two, namely interviews with development staff and examination of documents. This project was also used by MJUK personnel to attempt to estimate cost using the 1988 Rollo and Ratcliff proposals described in Section 3.3. The JSD-FP counts obtained from this project are as given in Table 6.3 below

Table 6.3

Project Name: Sharelink

States (S)	127
Transitions (T)	141
Input Types (IT)	339
Output Types (OT)	336
State Vectors (SV)	73
Data Streams (DS)	70
Total JSD-FPs	**1086**

SMCS Oceanography

This project was a part of the Submarine Command System being developed by SEMA Scientific on behalf of the Royal Navy. Data for this project was collected by the staff of SEMA due to the confidential nature of the project; and because of this, the accuracy of the data cannot be guaranteed. The development was written in Ada code and the developers had experience of both JSD and Ada. The developers had access to the full range of JSD tools. The JSD-FP counts obtained from this project are included in Table 6.4.

Table 6.4

Project Name: Oceanography

States (S)	55
Transitions (T)	62
Input Types (IT)	117
Output Types (OT)	762
State Vectors (SV)	68
Data Streams (DS)	84
Total JSD-FPs	**1148**

SMCS Position Handling

This was another phase of the Submarine Command Systems Developed by SEMA Scientific. The data was collected by SEMA staff and the full range of tools was also available. The JSD-FP counts obtained from this project are given in table 6.5 below.

Table 6.5 Project Name: Position Handling	
States (S)	30
Transitions (T)	39
Input Types (IT)	118
Output Types (OT)	48
State Vectors (SV)	49
Data Streams (DS)	60
Total JSD-FPs	**344**

Helicopter

This was a real-time project which was the first phase of a helicopter autopilot simulator being developed on behalf of the Defence Research Agency based at RAE Bedford. The project was developed by MJUK personnel who used a newly developed JSD tool called Adacode. This tool allowed the system to be developed by machine transformation directly from the JSD specifications. The development team consisted of two members of staff, one of whom was the developer of the new tool. The data was collected by visiting the site of

MJU K, interviewing the personnel and examining the documentation. The data resulted in the JSD-FP counts given in Table 6.6.

Table 6.6 **Project Name: Helicopter**	
States (S)	7
Transitions (T)	5
Input Types (IT)	6
Output Types (OT)	14
State Vectors (SV)	4
Data Streams (DS)	18
Total JSD-FP's	**54**

6.4 Analysis of Results

The data obtained for the projects given above was analysed by producing a correlation matrix. The correlation matrix is given in Table 6.7. The correlation matrix demonstrates that there are some variables which are multicollinear with each other, most notably states and transitions. The correlation between states and transitions is to be expected as each state must have a transition connecting it to another state. However, the presence of multicollinearity implies some difficulties with the model, especially if the model is based on multiple regression. The formula used to obtain the size calculations in JSD-FPs is not the result of multiple regression. However, the possibility exists that the model may be oversensitive, and it was decided therefore to run a Monte Carlo simulation to test the stability of the model with respect to variations in the input variables. This also seems advisable as, in the

case of two of the projects the data was collected by a third party and hence its accuracy was not known. In the case of data collected directly, there were a few cases where the full data was not available and had to be estimated by the analysts involved in the project. This was commonly the case where some documentation was missing or out of date.

Table 6.7 Correlation Matrix of Model Variables

	ST	TR	IT	OT	SV	DS
ST	1.0	0.9993	0.8431	0.8097	0.7989	0.8735
TR	0.9993	1.0	0.8370	0.8062	0.7866	0.8617
IT	0.8431	0.8370	1.0	0.6780	0.9639	0.9298
OT	0.8097	0.8062	0.6780	1.0	0.7147	0.7805
SV	0.7989	0.7866	0.9639	0.7147	1.0	0.9786
DS	0.8735	0.8617	0.9298	0.7805	0.9786	1.0

Simulation

The simulation was carried out using a product known as Crystal Ball (Decisioneering, 1993) which is an add-on to the Microsoft Excel spreadsheet. This product requires the user to decide upon the probability density functions that apply to the various variables in the model. The tool also allows the entry of any correlation between data items so that the data generated will maintain the characteristics of the actual data obtained.

The results of the simulation are presented below. An examination of the available data reveals that it has the statistical characteristics shown in Table 6.8. The simulation model was set up with assumptions that reflect the statistics

in Table 6.8, namely with the means as shown used as starting values, but with a minimum of zero and an undefined maximum (clearly no attribute can have less than zero occurrences, whereas the upper-bounds may well be very large).

Table 6.8 Statistics Associated with Project Data

Attribute	Mean	Std Dev.	Minimum	Maximum	Label
ST	112.17	99.13	7	254	states in JSD entity
TR	121.50	104.13	5	267	transitions in a JSD entity
IT	278.33	313.77	6	877	input types
OT	491.50	419.01	14	1019	output types
SV	143.83	206.31	4	559	state vectors
DS	171.33	208.38	18	572	data streams

The data for the simulation represents the number of occurrences of the various attributes in a single specification. Thus the data is best represented by a Poisson distribution, as this distribution best describes the number of occurrences in a given interval such as the number of errors in a page. The results of the simulation are shown as a forecast chart (Fig 6.1) depicting the distribution of the number of JSD-FP's that were generated by the simulation.

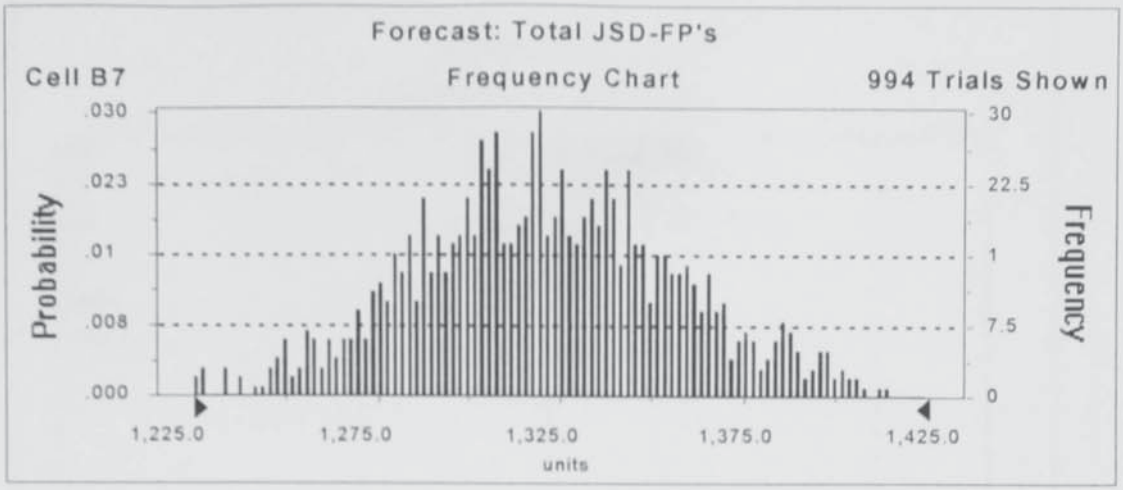


Figure 6.1 Distribution of Total JSD-FP's generated by the simulation

The sensitivity analysis is shown in Figure 6.2 and depicts the contribution to variance of each of the attributes of a JSD specification obtained from the simulation. As discussed earlier, there is a very high correlation between states and transitions in a JSD specification; the simulation was therefore run with the variables ST and TR constrained to exhibit the correlation obtained by analysis of the original data, namely 0.9993. Simulations were run with correlation present and then without correlation, in both cases, the contributions to variance were the same value which is shown in Figure 6.2. Correlation was not used between any other variable because while one would expect all variables to increase in value as a system gets larger there is no defined relationship as exists between states and transitions, it is, for example, perfectly possible for a system to have a high number of inputs and a low number of outputs.

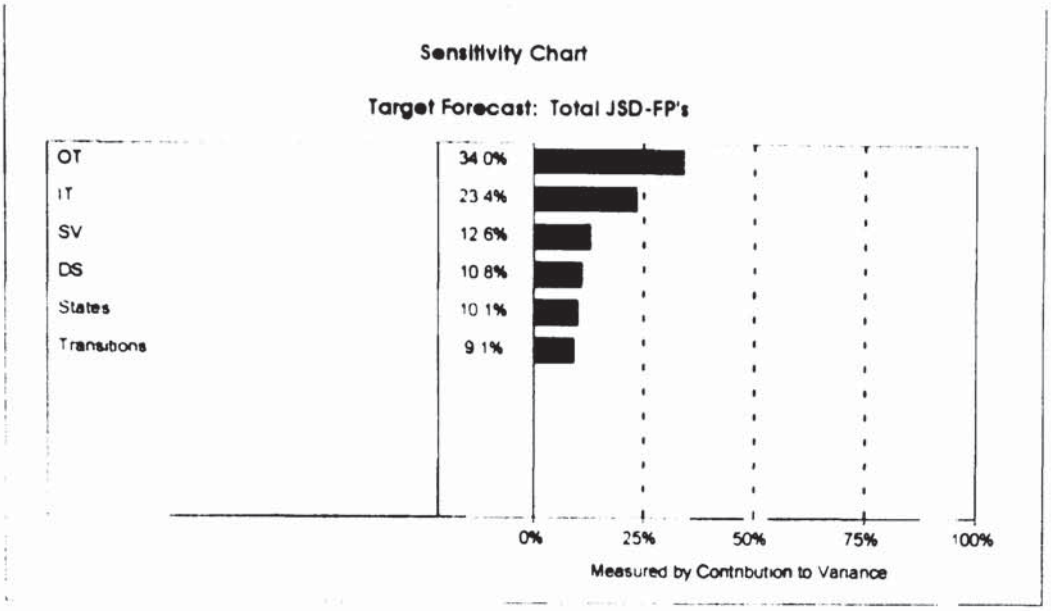


Figure 6.2 Sensitivity Analysis of Simulated Variables

The sensitivity analysis shows that the greatest contribution to variance is from the OT and IT attributes of a specification, as OT and IT are the variables that generally occur in the largest number and as we are dealing with a simple additive model, this is to be expected. The implication is that when estimating JSD attributes, particular care should be taken with the estimates of OT and IT parameters.

6.5 Estimating Using JSD-FP Counts

The generation of a regression formula relating the JSD-FP size to the effort used was not undertaken, as there was insufficient data available to undertake this form of modelling. Instead, the approach taken by many current methods was used — namely, the derivation of productivity figures based upon projects undertaken by the same development team on similar projects. For the Sharelink project, the productivity figure obtained for Cheques was used as it is a similar project. For the Cheques, Oceanographic and Helicopter projects, these being the first of their type in the development environment, productivity figures were not available.

Table 6.9 Results of predictions using JSD-FP's and Productivity					
System	**JSD-FPs**	**Work Days**	**Productivity**	**Forecast work days**	**Error**
Cheques	3548	3640	0.974725		
Pensions	1732	1319	1.313116	1776.911	1.34
Sharelink	1086	927	1.171521	1114.16	1.28
Oceanographic	1148	3832	0.299582		
Position Handling	344	1275	0.269804	1148.265	0.90
Helicopter	54	59	0.915254		

These results seem somewhat disappointing at first glance. However, there was an expected increase in productivity between Cheques and Pensions, because of the development teams increasing familiarity with the methods and tools being used. In a sense the Sharelink figures are misleading as the project, while similar in nature to Cheques and Pensions, was undertaken in a different development environment. The forecast for Position Handling is an underestimate by some 10%. This project had a somewhat lower productivity rate than was obtained for Oceanographic. However Oceanographic had a larger database element and was therefore to some extent a hybrid project of real-time and data processing, and this may explain the difference in productivity. Allowing for these caveats, it would seem that the forecasts produced using development environment productivity figures are reasonable. It should be remembered that when a project is being undertaken in a new tool and method environment, the CCTA suggests that developers should allow for an increase in development effort of up to 35%, for the first project and 20% for the second. If due allowance was made for this effect the productivity for Cheques would be adjusted to 1.22 JSD-FP/day (using 20%), which would have given a forecast of 1421 work days for Pensions an error of 1.08 or an overestimate of some 8%. This effect is noted by Boehm (1988) who shows that for Ada projects the

development team require five projects to achieve their previous productivity rate.

After the consideration above it can be seen that the results are in fact quite encouraging and that they support the findings of other authors. From the point of view of the practising manager or estimator in industry, the accuracy obtainable using the data from one completed project obviates the need for the analysis of large amounts of historical data. Hence an estimating programme may be instigated on the basis of one previous project, plus an understanding of the advice given by various authorities. The figures obtained bear out the advice given by the CCTA (1991) in that initial use of a new method will reduce productivity. As shown once this effect is allowed for, the estimates for Position Handling (10%) and Pensions (8%) are well within the accuracy that might be expected using the industry standard figures for FPA published by Symons (1991). The results lend further support to the idea of estimation by analogy (Cowderoy, 1990); we see that grouping the Sharelink project with other similar projects gave an acceptable result. The Sharelink project is analogous to Pensions and Cheques in that they are all financial sector projects developed using the same toolsets.

The results of this research also support the findings of the MERMAID project, which showed that the use of small data sets was quite viable. Taking this a stage further it is possible to assert that in many situations small data sets are to be preferred to large data sets. This is especially true where the small set is homogeneous, since a large set of data from several projects is very likely to be heterogeneous. That is, with a large data set the development environment will tend to differ across the range of projects. An illustration of this is the data set used by Boehm (1984) which reveals a large variation in productivity.

However if the data is grouped according to the factors shown in Table 6.10, this variation is reduced. For example in the case of organic mode scientific applications of similar size (in KDSI) and written in FORTRAN under the same environmental factors (TOOL and MODP being similar - between 1.1 and 0.95) the range of productivity becomes 302 - 723 actual DSI per man month, rather than 47 - 1250 DSI/MM when the full range of scientific applications is considered. The point to note is that by using a small heterogeneous data set then variety is reduced and forecasts may be made with more confidence.

Table 6.10 Data Extracted From the COCOMO Data Set

All Scientific Applications (Boehm, 1981. pp 496, 497)

Project number	Year	Lang.	MODP	TOOL	Mode	Size KDSI	Prod DSI/MM
31	75	MOL	1.0	1.0	E	50	47
32	72	FTN	1.10	1.10	SD	261	372
33	76	FTN	0.91	0.91	E	40	66
34	77	FTN	0.91	1.10	E	22	66
35	68	MOL	1.24	1.24	E	13	159
36	79	FTN	1.0	0.91	SD	12	218
37	78	FTN	1.0	1.0	ORG	34	723
38	77	PL1	.82	.91	ORG	15	1250
39	64	FTN	0.91	1.10	ORG	6.2	775
40	74	MOL	1.10	1.0	ORG	2.5	312
41	76	FTN	0.91	0.91	ORG	5.3	883
42	78	FTN	0.95	0.95	ORG	19.5	433
43	78	FTN	1.0	1.0	ORG	28	337
44	78	FTN	1.10	1.0	ORG	30	345
45	77	FTN	1.0	0.95	ORG	32	302
46	78	FTN	1.0	1.0	ORG	57	452
47	78	FTN	1.10	1.0	ORG	23	639

6.6 Conclusion

This chapter has presented the results of using the Albrecht & Gaffney based counting rules in industry and the problems which were encountered. The JSD-FPA model has also been analysed and the results presented. The results of the analysis demonstrate that the JSD-FPA method is a viable technique for estimating a JSD-based project at an early life-cycle stage. The method will also allow estimates to be refined as the development of the JSD specification proceeds and estimates of attribute values can be replaced by the actual values. The results support the advice of the CCTA (1991) that first use of method and tools initially lowers productivity. They also demonstrate that the findings of the MERMAID project concerning the use of small data sets are sound and that estimation by analogy can be used to reduce the variety in past project data.

Chapter 7

Further Discussion and Conclusions

Introduction

This chapter will recapitulate the importance of the research topic, summarise the main points of the approach that was taken and highlight the main recommendations as to the estimating practice that should be followed by software developers operating within the context of the JSD method. The chapter will conclude with suggestions for further research that will take the topic forward.

7.1 Recapitulation and Discussion

The discipline of software engineering is still immature; much software development is still carried out in an *ad hoc* manner. If software engineering is to become a mature discipline worthy of being classed as 'engineering', then its practitioners require the development of a range of measures, standards and practices, in various areas, which will allow software engineers to improve the efficacy of the processes and the quality of the resulting products. One such area is that of software cost estimation. Several methods have been proposed since the early 70's which attempt to provide a standard estimating technique; all have met with some success in certain contexts but none has so far proved to be universally applicable.

The area of software cost, or effort estimation, remains one where it is still difficult for the practising engineer to make confident predictions. The major difficulties in the development of standard estimating techniques are concerned with the wide variation in: application domains, project development environments, and the number of factors which have a bearing on the final project costs. For these reasons most of the current cost estimating methods recommend calibration within the local development environment. While this goes some way to allowing for the development environment, many methods still attempt universal, or near universal, applicability. The underlying assumption that there exists a technique which will produce an accurate estimate of costs in any situation has been challenged by the development of techniques — such as the method proposed as an outcome of the MERMAID project (Van Riet, 1992) — which are solely concerned with the particular environment of the project being estimated. A method which has shown promise in use, Function Point Analysis, is only claimed to be useful in the domain of commercial information systems development. Recently, however the increasing sophistication and complexity of some commercial systems, as well as the needs of the real-time and scientific systems developer, has created interest in expanding the applicability of this technique.

The topic of this research has been provoked by considerations such as those outlined above. Firstly, that it is unlikely, given the variety in the field of software development, that a single universally applicable method exists or could be developed. Secondly that while it is possible to develop a set of estimating procedures completely tailored to specific needs and environments, as with MERMAID there is a need for developers and their customers to be able to draw comparisons and conclusions across project teams and between different organisations. Thirdly, it is desirable to make any method developed accessible to the practising software engineer without the diversion of effort

that would be required to carry out either extensive local calibration or the development of a purely local forecasting technique. The research has therefore concentrated on the development of a sizing and estimating technique around a single software development method, namely the Jackson System Development method. In this way variety could be reduced, but the estimating technique would be applicable to a sufficiently wide range of projects to allow useful comparisons to be made of, for example, productivity in different environments.

7.2 Summary of the Approach

The approach taken has been to examine the existing range of methods in order to determine to what extent they could be adapted to provide a standard method which could be used in the context of a JSD project. In order to achieve this, an extensive survey of the current literature was undertaken; this revealed that there were a small number of methods which have gained acceptance in the computing community. These methods were then explored in order to reduce the number to be considered in detail. The outcome of this was that two main methods, COCOMO and FPA, were then considered in the context of JSD. FPA has two main variants which were both considered.

The principal difficulties in applying FPA techniques directly to a JSD project were found to be those of correctly interpreting the counting rules in a JSD environment. Mappings between JSD and FPA for both the Albrecht & Gaffney and the later MKII version of FPA were developed. The use of these mappings on actual projects showed that, while FPA is usable in a JSD, context its use is restricted to those projects being undertaken in the area of conventional commercial information systems. In the case of commercial information systems MKII FPA can be applied directly though it will require

analysis effort, identifying transactions and some entity relationship modelling, which departs from the standard JSD approach. Alternatively the counting rules of Section 3.3 can be used. However for projects with a substantial element of real-time processing none of these methods will give reliable results. The MKII variant of FPA was used as part of the inspiration for the development of a new metric and accompanying counting practice guidelines, provisionally called JSD-FPA. This technique is based upon the characteristics of a JSD specification, such as data streams, state vectors, and input and output data, and may therefore be carried out as part of the normal process of analysis. The technique does not require the specification to have been developed in every detail and may therefore be used at a sufficiently early point in the life-cycle to provide estimates for costing the project before contracts are finalised. It is normal for the preliminary analysis phase to produce sufficient detail so that estimates of JSD specification characteristics may be produced. As the project proceeds, these estimates should be revised in the light of actual data.

The method COCOMO uses the number of expected delivered source instructions as a system size metric. A problem, which arises when using DSI at a sufficiently early life-cycle point to be useful to the estimator, is that of producing accurate estimates of DSI. As a JSD project progresses through the development stages, it becomes feasible to consider an estimating approach based on later life-cycle deliverables. Therefore, a method of sizing the expected source instructions from early life-cycle products, was developed. This sizing technique is used as a front end to conventional COCOMO to give an estimating technique which was called JSD-COCOMO.

The development of complementary approaches (i.e., one FPA-based and one SLOC-based) allows the software developer to make two independent

estimates, and thus improve the confidence in the estimate and narrow the estimation interval. The overall approach developed therefore provides several stages in the development at which different estimation techniques are employed to refine the estimates as work progresses. The developed method has been validated against several projects and by the use of simulation. While it has to be accepted that validation was made against insufficient projects to give complete confidence, nevertheless the sizing method proposed as an alternative to the MKII method (JSD-FPA) shows promise.

7.3 Critical Review and Statement of Further Work Needed

The work that has been undertaken in conducting the research has led to the development of proposals for the sizing and estimating of JSD projects. When this account of the work is reviewed several strengths and weaknesses are evident. The chief strength of the research — the determination to base the research on 'real' projects as opposed to student projects — is also the root of the major weakness — the small number of projects available on which to base a validation. It has proved very difficult to obtain free access to project data, and gratitude is due to those organisations which allowed access, or provided the data from their own resources. However, the small number of projects available for validation mean that the work as presented cannot be considered to be of a fully validated set of estimating models. Fortunately sufficient data is available so that this account of the work has demonstrated the viability of the JSD-FPA approach, though no claims can be made for the accuracy of this model. The JSD-COCOMO model suffers to an even greater extent from the lack of data, the viability of the sizing metric is certainly demonstrated although this cannot be said to have been adequately demonstrated in the case of that

metric combined with COCOMO to make up the JSD-COCOMO model. These considerations lead on to the conclusion that an area where further work needs to be done is in the validation of the models, it has to be said that indications from the industry are that considerably more interest exists for the JSD-FPA model than for JSD-COCOMO. Certainly the findings of this work, along with an offer to support the introduction of the models, will be disseminated to the JSD community, and should the interest shown be pursued by any organisation then sufficient data to validate the model will probably become available.

The JSD-FPA and JSD-COCOMO models make no attempt to provide estimates for the preliminary phases of software development.. The phase breakdown in Table 3.3 does not provide a figure for the project initiation stage of JSD. This is contrary to the practice in COCOMO where a figure of some 6% of total development effort is suggested (thus giving a total of 106% !). Other estimating models attempt to quantify this phase and therefore this may be seen as a weakness. This is a deliberate position which is taken following conversations with developers, IT planners and managers during which the impression gained is that the preliminary phase varies considerably in the amount of effort expended. Some companies conduct very detailed and thorough business investigations alongside technical feasibility studies other organisations are content provided development fits their strategic direction and the cost falls within that fraction of their budgets which they consider sufficient for new IT development. A further reason is that the work in this phase has already been undertaken and so providing an estimate for it seems a pointless exercise, especially given the probable inaccuracy. To some extent there is an element of presentation in this decision — if an estimate is made which has a significant deviation to the actual, for a phase that is complete or very nearly so, then the credibility of the overall estimate will come into question, yet, there is

very little evidence that a sensible costing model could be developed that applies across several organisations to this aspect of a software or systems project.

7.4 Further Research Topics

As part of the research undertaken when investigating the FPA methods, a series of experiments was mounted (Rollo & Steven, 1993) to establish the inter-counter and inter-method reliability of the two FPA variants (Albrecht & Gaffney, 1983; Symons, 1991). Apart from the main result of the experiment — establishing the reliability of the two methods — two other issues emerged: firstly that it appears possible to apply both methods at a very early life-cycle point. Given only a users' statement of requirements, a group of computer science students were able to produce estimates, the mean of which agreed with those produced from the full specification, albeit with a wide degree of variation; they also had an inter-counter reliability at the 95% level of confidence. The accepted point at which FPA may be applied is after the analysts have undertaken preliminary feasibility studies and produced a detailed requirements specification (CCTA, 1991). The function point counters in these experiments were undergraduate students, and it seems entirely reasonable to suppose that more experienced analysts working with real projects might produce results with a lower variance. In order to establish this, it would be necessary for further research to be undertaken using experienced analysts and with real projects.

The second issue thrown up by these experiments concerns the relationship between the two function point techniques. The Rollo and Steven experiment utilised a case study which had been previously rated at 140 FP. The estimates produced by the student counters showed that MKII FPA was providing a

significantly lower estimate than the original method at this point. Symons (1991) states that there is no linear relationship between the MKII method and the original, but that MKII FP was calibrated to agree with the original at 1000 function points, and that MKII would produce a higher estimate above this figure. The assertion by Symons that there is such no relationship was made in the light of a comparison of a small number of systems. The UFPUG counting practices manual (UFPUG, 1994) asserts that there is no "simple relationship between the two methods" but that there is a roughly linear relationship up to about 400 function points. No evidence is provided for these assertions which are, presumably, based on the experience of the counting practices committee. A consequence is that several practitioners of function points utilise the language coefficients developed for the Albrecht & Gaffney method with the MKII method, and clearly this may be an inappropriate use of these figures. Research is needed which establishes the relationship between the two methods and, if possible, a sound way of converting between them; this would be of use to those developers who wish to migrate to the MKII method, and *vice versa*. It would also be useful for estimators who wish to undertake estimates of SLOC given that the relationship between the MKII method and SLOC in various languages could be established.

Technical Complexity Factors

The use of Processing Complexity Adjustment, as it is are known in the original FPA method, or Technical Complexity Adjustment, as Symons called it in his MKII method, is an area which would benefit from further research. In his book, Symons (1991) produces a critique of PCA, yet goes on to adopt all of the factors and propose five more. Several authors have shown that the use of these factors has no effect on the accuracy of the final estimate (Betteridge, 1992; Heemstra & Kusters, 1991), and indeed in one such study the accuracy was reduced (Kemerer, 1987). An alternative approach has been suggested by

Rule (1993) in which any adjustment should be based upon the *non-functional requirements* of the system. This is an appealing approach, though as yet the idea is based on a subjective assessment of the extra effort required to develop a system because of its non-functional requirements. The need is for some research to attempt to develop a technique which will allow the extra effort to be calculated, possibly based on the amount of the difference in the level of a non-functional requirement as compared to the normal level in typical projects in that environment (i.e., x% improvement means y% extra effort).

Risk Analysis and Management

Alongside cost estimations the software engineer also needs some insight into the risks associated with whether or not the estimate will be met. Most cost estimating techniques produce single-point estimates and do not reflect the uncertainty in that estimate, even less do they allow for 'what if' questions to be explored. The MKII method merely makes assertions of the type that if this is the "first use of new method" then the estimate should be increased by 33% (CCTA, 1991). The COCOMO method has spawned the development of a COCOMO-based risk assessment technique (Fairley, 1991). The development of a similar technique tailored to suit the function-based methods would be a useful addition to the software engineer's tool kit.

Object Orientation

The JSD method has some features in common with certain object oriented methods being used increasingly. JSD is not an object oriented method but could be described as 'object based', in that both object oriented methods and JSD start by considering entities or objects in the real-world environment of the system to be developed. The proposed method, JSD-FPA, could therefore have

some applicability for object oriented methods. The applicability should be investigated, and depending upon how well-matched the technique is for an object oriented method, an object oriented variant of JSD-FPA could be developed.

Graphical User Interfaces

There is disagreement within the function point community as to how one should count a graphical user interface (GUI). Disagreement centres around to which extent the GUI represents a functional requirement and hence, for example, which aspect of a window should be counted. Other practitioners feel that the GUI should be treated merely as an extension of the non-functional requirement in the usability category. The UFPUG counting practices committee has developed an early draft of guidelines to assist FP analysts in counting such interfaces. There is a need for some research to establish how best to count the features of a GUI, and to produce a set of counting rules which could then be incorporated in the various function-based methods.

Tool support for JSD-FPA

From the outset, it was an aim of this research to develop an estimating method for JSD which would facilitate the *automatic* calculation of system size from the data produced by the development process. This aim was realised, and therefore it is possible to carry out system sizing automatically directly from the database contents of the CASE tools that are employed by JSD developers. Due to the demise of Michael Jackson Systems Ltd, who merged with Learmonth and Burchett Management systems, support for JSD CASE tool development has reduced to the point where several JSD users have now produced their own tools. Consequently, a tool add-on was not developed as

part of this research. Nevertheless, the findings of this research will be disseminated via the Jackson User Group, which is still extant, so that JSD developers may consider incorporating the estimation techniques into their own tools.

7.5 Conclusion

This chapter has recapitulated the need for the research and given an account of the approach which was used. The research has been reviewed and its chief weaknesses and strength highlighted. Several topics for further research, emerged during the conduct of this research and these have been detailed. The research described has led to the development of metrics which form the basis of two approaches to estimating in a JSD environment. The JSD-FPA metric has been shown to offer a promising method for analysts working within JSD developments to undertake the estimating of project effort and cost. The JSD-COCOMO method is proposed as a bottom-up technique which can be used to reinforce the function based estimates.

REFERENCES

Abdel-Hamid, T.K. & Madnick, S.E. (1986) 'Impact of schedule estimation on software project behaviour', *IEEE Software*, **Jul.**, 70-75

Albrecht, A.J. (1979), 'Measuring application development productivity' *Proceedings Jint SHARE GUIDE IBM Application Development Symposium*, October 1979 pp 83-92.

Albrecht, A.J. & Gaffney, J.E., Jr. (1983) 'Software function, source lines of code, and development effort prediction: A Software Science Validation', *IEEE Transactions on Software Engineering*, **SE-9**(6), 639-648.

Ami. (1992) *The Ami Handbook*, CSSE Southbank Polytechnic 1992, London.

Amos, C. (1993), 'Software acquisition reducing the risks,' *Proceedings ESCOMM*, 1993

Amos C., (1993) "*Software Acquisition - Reducing the Risks*" Proceedings of the European Software Cost Modellers Meeting, Bristol 1993.

Arunkumar, S. & Lee, S.H. (1979) 'Enumeration of all minimum cut-sets for a node pair in a graph', *IEEE Transactions on Reliability*, **R-28**(1), 51-55.

Bailey, JAW. & Basili, V.R. (1981) 'A meta-model for software development resource expenditures', *Proceedings, Fifth International Conference on Software Engineering*, IEEE/ACM/NBS, March 1981, pp. 107-116.

Balzer, R., Goldman, N. M., & Wile, D. S. (1982) 'Operational specification as the basis for rapid prototyping', *ACM SIGSOFT Software Engineering Notes*, 7(5).

Baskette, J. (1987) 'Life-cycle analysis of an Ada project', *IEEE Software*, **Jan** 1987, 40-47.

Bass, A. P. (1992) *The transformational implementation of JSD process specifications via automata representation*, PhD Thesis, Aston University Aston Triangle Birmingham UK.

Betteridge, R., Fisher, D., and Goodman, P., 'Function Points Vs. lines of code', *System Development*, August 1990.

Betteridge, R. (1992), 'Uses and abuses of function point analysis'. *European Function Point User Group Conference*, London, June 1992.

Black, R.K.D et al (1977), *BCS software production data*, Final Technical Report, RADC-TR-116, Boeing Computer Services, Inc., March 1977. NTIS No. AD-A039852.

Boehm, B.W. (1981) *Software engineering economics*. Prentice-Hall, Englewood Cliffs, N.J.

Boehm, B.W. (1984) 'Software engineering economics', *IEEE Transactions on Software Engineering*, **SE-10**(1), 4-21.

Boehm, B.W. (1987) 'Improving software productivity,' *IEEE Computing 198? pp 43-57*.

Boehm, B.W. (1988) 'A spiral model of software development and enhancement', *IEEE Computer*, May 1988, 61-71.

Boehm, B.W. (1988b) 'Ada COCOMO refinements', Fourth COCOMO Users Group Meeting held at the Software Engineering Institute, Carnegie Mellon University, Pittsburgh Pennsylvania.

Boehm, B.W. (1989) 'Theory - W software project management: principles and examples'. *IEEE Transactions on Software Engineering*, **SE-15**(7),902-915.

Booch, G. (1983) *Software engineering with Ada* , Benjamin Cummings, Menlo Park, California.

Bowerman, B.L. & O'Connell, R.T. (1990), '*Linear statistical* models*: an applied approach*', PWS-KENT Publishing Company, Boston.

Bowman, M. & Butcher, M (1986) 'JSD at Marconi underwater systems'. *Jackson user update* 1986.

Brooks, F.P. Jr. (1975) *The mythical man month,* ' Addison Wesley, Reading MA.

Business. (1991) The quality imperative, *International Business Week* December 2 1991, pp 18-83, McGraw Hill

Butcher, M & Laddington, M (1987) 'Applications of JSD for real-time systems - news, reviews & project updates' *Jackson user update* 1987.

Callisen, H. Colborne, S. (1984), 'A proposed method for estimating software cost from requirements'. *J Parametrics*, **4.4** December 1984.

Cameron, J. (1986) 'An overview of JSD', *IEEE Transactions on Software Engineering*, **SE-12**(2), 222-240.

Cameron, J. (1988) 'The modelling phase of JSD', *Information and Software Technology*, **30**(6), 373-383.

Carriere, W.M. Thibodeau, R. (1979), 'Development of a logistics software cost estimating technique for military foreign sales' *Report CR-3-839, General Research Corp.* - Reported in Boehm (1981).

CCTA (1991). *'Estimating with MKII function point analysis'* SSADM Version 4 Subject guide, CCTA publication HMSO London.

Conte, S.D. , Dunsmore, H.E. & Shen, V.Y. (1986) *Software engineering metrics and models*, Benjamin Cummings publishing company, Menlo Park, California.

Cowderoy, A. (1990) 'AMY-1 An analogy-based estiMation sYstem' (ESPRIT project p2046), European COCOMO Users Group Meeting Botley Hampshire, England, May 1990.

Cuelenare, A.M.E Van Genuchen, M.J.I.M and Heemstra, F.J. (1987) 'Calibrating a software cost estimation model: why and how', *Information and Software Technology*. Vol. 29 p558.

Decisioneering (1993), *Crystal Ball version 3.0 user manual*, Decisioneering Inc., Denver Colorado.

Dedene, G. (1989) 'Information and function points in JSD', *Jackson user group conference*, Jac89, Eastbourne.

DeMarco, T. (1982) *Controlling software projects*, Yourdon press - Prentice-Hall, Englwood Cliffs, N.J.

Dhal, O.J., Dijkstra, E., & Hoare C.A.R. (1972) *'Structured programming'*, Academic Press, 1972

Dreger, J. B. (1989) *'Function point analysis'*, Prentice Hall, Englewood Cliffs, New Jersey 1989.

Duncan, A.S. (1988) 'Software development productivity tools and metrics', *IEEE Software*, 1988, 41-48.

Fairley, R.E. (1991) 'An overview of risk analysis using cost models' *European software cost modelling meeting 1991*, ESTER, Noordwijk, Netherlands.

Feather, M. S. (1982) 'Mappings for rapid prototyping', *ACM SIGSOFT Software Engineering Notes*, 7(5), 17-24.

Ferrentino, A.B. (1981) 'Making software development estimates "good"', *Datamation*, Sept. 1981, 236- 238.

France, N. & Lawton, J. (1987) 'JSD used on a command & control project' *Jackson User conference* 1987.

Fenton, N.E. (1991) *Software Metrics - a rigorous approach*, Chapman & Hall London.

Freiman, F.R. & Park, R.E. (1979) 'PRICE software model - Version 3: an overview', *Proceedings, IEEE-PINY Workshop on Quantitative Software Models*, IEEE catalogue No, TH0067-9, October, pp 32-41.

Gallacher, J. (1984) 'Project estimation and control' *In - Software engineering for microprocessor systems* - ED Depledge, P, Peter Perigrinus Ltd, London.

Hall, N.R. (1983) '*Complexity measures for systems design*', PhD Thesis, Polytechnic Institute of New York.

Heemstra, F.J. & Kusters, R. J. (1991) 'Function point analysis: evaluation of a software cost estimation model' *European Journal of Information systems,* **1** 229-237

Herd, J.R. (1977) *Software cost estimation study - study results,* Final Technical Report, RADC-TR-77-220, Vol. 1, Doty Associates, Inc., Rockville, MD, June 1977.

Hughes, R. (1994) 'An exploration of the Problems of Using Function Points MKII', *European Journal of Information Systems*, **4** 169-183
ISQAA (1989) '*Metrics Handbook*', Information systems quality assurance association, 1989

IFPUG (1988), '*Function point counting practices manual,*' Version 3.4, International Function Point Users Group, July 1988.

IFPUG (1992), '*Function point counting practices manual,*' Version 3.4, International Function Point Users Group, July 1992.

Ince, .D (1989) 'Software metrics - an introduction', Seminar series on new directions in software development, Wolverhampton Polytechnic may 1989.

ISO (1994), 'Draft ISO/IEC Standard - Information Technology - Software measurement - Definition of Functional Size Measures, Version 1.3 June 16 1994.

Jackson, M. A. (1983) *System Development.* Prentice-Hall, Englewood Cliffs, N.J.

Jensen, R.W. (1980) 'An approved microlevel software development resource estimation model', *14th Asimolar conference on circuits, systems and computers,* Pacific Grove California, Nov. 1980.

Jones, C. (1986) *Programmer productivity,* McGraw-Hill, New York.

Jones, C. (1987) 'Function Point metrics: key to improved productivity', *Information week,* **Feb. 23,** 26-27.

Jones, C. (1988) *A short history of function points and feature points*', Software Productivity Research, Inc., Cambridge, MA.

Kemerer, C.F. (1987) 'An empirical validation of software cost estimation models', *Communications of the ACM,* **V30**(5), 416-429.

Kemerer, C.F. (1990) '*Reliability of function points measurement: a field experiment*', Working paper MIT Slogan School of Management WP#3193-90-MSA, Massachusetts Institute of Technology, Cambridge MA, 1990

Kitchenham, B.A. Linkman S.J. (1989), '*Design metrics in practice*', Seminar series on new directions in software development. Wolverhampton Polytechnic, may 1989

Kitchenham, B., (1991) 'Metrics for project control', Proceedings ESCOMM, 1991, ESTEC centre, Noordwijk, Netherlands.

Kitchenham, B., (1991) '*Making Process Predictions*' in Software Metrics Fenton N ed., Chapman & Hall London 1991.

Kunkler, J.E. (1985) 'A co-operative Industry Study: Software Development/Maintenance Productivity' Xerox Corporation March 1985.

LBMS. (1991*), Introduction to LBMS Jackson System Development (JSD) version 2.0*, Learmonth and Burchett management Systems Plc, London 1991.

Li, H.F. & Cheung, W.K. (1987) 'An empirical study of software metrics', *IEEE Transactions on Software Engineering*, **SE-13**(6), 697-707.

Lind, R.K. & Varian, K. (1989) 'An experimental investigation of software metrics and their relationship to software development effort', *IEEE Transactions on Software Engineering*, **SE-15**(5), 649-653.

Londeix, B. (1987) *Cost Estimation for Software Development*. Addison-Wesley, Wokingham, Berks.

Low, G.C. & Ross Jeffrey, D. (1990) 'Function points in the estimation and evaluation of the software Process', *IEEE Transactions on Software Engineering*, **SE-16**(1), 64-71.

Macro, A. Buxton, J (1987) '*The craft of software engineering*', Addison Wesley, Wokingham England.

Mallinson, D.B. (1987) 'The humble art of estimating', *ASM Journal of Systems Management*, DEC 1987, 23-35.

Masters, T.F. (1985), 'An overview of software cost estimating at the NSA. *Journal of Parametrics*, **5.1** Mar 1985 pp 72-85.

Marouane, R. Milli, A (1989) 'Economics of software project management in Tunisia: Basic TUCOMO.' *Information and Software Technology*, **V31**(5) June 1989 pp 251 -267.

Martin, R. & Macdonald, D. (1988) 'Evaluation of current software costing tools', *ACM SIGSOFT Software Engineering Notes*, **V13**(3), 49- 51.

McCabe, T.J. (1976) 'A complexity metric', *IEEE Transactions on Software Engineering*, **SE 2**(4).

McTavish, D.I. (1984) 'Guidelines for the establishment of operational software teams', RIU/501/3, RAE Malvern.

McNicholls, D.G. (1982) *Predictive model of resource consumption during the program development phase of software development*, PhD Thesis, University of Missouri - Rolla.

Mellichamp, D.A., Ed (1983) *Real-Time Computing with Applications to Data Acquisition and Control*, Van Nostrand Reinhold, New York, 1983.

Miyazaki, Y. & Moro, K. (1985) 'COCOMO Evaluation and tailoring', *IEEE Software*, *pp 292-299*.

MJUK (1989a), *Task based estimation, Technical Report*, Michael Jackson (UK) Ltd, MJUK Technology House, Langley, Berks.

MJUK (1989b), *Estimation for JSD projects, Technical Report*, Michael Jackson (UK) Ltd, MJUK Technology House, Langley, Berks.

MJUK (1990), *An Analysis of the effectiveness of in-house estimating techniques used within MJUK, Technical Report*, Michael Jackson (UK) Ltd, MJUK Technology House, Langley, Berks.

Mohanty, S.N. (1981) 'Software cost estimation: present and future', *Software Practice and Experience*, **11**(2), 103-121.

Mukhopadhyay, T and Kekre, S. (1992) 'Software effort models for early estimation of process control applications' *IEEE Transactions on Software Engineering*, **SE 18**(10), 915-924

Myers, W. (1989) 'Allow plenty of time for large-scale software', *IEEE Computer* , Jul. 1989, 93-99.

Naur, P., Randell, B. & Buxton, J.N. (Eds.) (1976) *Software engineering concepts and techniques.* Petrocelli/Charter New York.

NCC. (1990) '*SSADM Version 4 Reference Manual*', NCC Blackwell Ltd, 1990.

Nelson, R. '*Software data collection and analysis at RADC*', Rome Air Development Centre, Rome, N.Y.

Norden P.V. (1958) '*Curve fitting for a model of applied research and development scheduling*' IBM J Vol. 2 No 3 Jul. 58.

Norden P.V., (1970) 'Useful tools for project management'. *in Management of Production*, Starr M.K., Ed. Penguin Baltimore pp 71 - 101

Perlis, A.J., Sayward, F.G. & Shaw, M (1981) *Software metrics, an analysis and evaluation*, MIT press, Cambridge, MA.

Pinsky, S.S (1984), 'The effect of complexity on software trade-off equations', Journal of Parametrics, 4 4 December 1984 pp 23-32.

Prather, R.E. (1984) 'An axiomatic theory of software complexity measure', *Computer Journal*, **V27**(4), 340-347.

Pressman, R.S. (1987) '*Software engineering: a practitioners approach*, McGraw Hill, New York, 1987

Pressman, R.S. (1990) '*Software engineering : a practitioners approach Vol. 3* ' McGraw Hill, New York, (1990).

Putnam, L. H. (1978) 'A general empirical solution to the macro software sizing and estimating problem', *IEEE Transactions on Software Engineering*, **SE-4**(4), 345-361.

Putnam, L.H. & Fitzsimmons, A. 'Estimating software costs', *Datamation*, September 1979, pp 189-198 continued in *Datamation* October 1979, pp 171-178 and November 1979, pp 137-140.

Ratcliff, B. (1987) *Software engineering principles and methods*. Blackwell Scientific Publications Oxford.

Ratcliffe, B & Rollo, A.L. (1990) ' Adapting function point analysis to Jackson Systems Development, *IEE Software Engineering, Journal*, January 1990 pp 79-84

Reifer, D.J. (1987) *Asset R: A function point sizing tool for scientific and real-time systems*, Riefer Consultants Inc., Torrance, CA.

Rollo, A.L. (1991) 'Jackson Systems Development', IEE Colloquium, Introduction to structured methods, London 1991.

Rollo, A.L. (1994) Applying Function Point Analysis to Real-Time Systems, A Seminar delivered to the Staff and Students of the Department of Computer Science The University of Tasmania, Hobart Australia.

Rollo, A.L. & Ratcliff, B. (1988) 'Software cost estimation for JSD projects - an interim proposal', *Jackson User Update 1988*, London.

Rollo, A.L. & Ratcliff, B (1990) 'Function Point Analysis and Jackson systems development' *European COCOMO User's Group Meeting 1990*, Hampshire.

Rollo, A.L. & Ratcliff, B (1992) 'Parametric sizing of software components at an early life-cycle Stage within the JSD Method. *European COCOMO User's Group Meeting 1992*, Munich.

Rollo, A.L. & Steven, C (1993) 'Function points reliability - A laboratory experiment, *European Software Cost Modelling Meeting 1993*, Bristol.

Ross, D. (1977) 'Structured Analysis (SA): A language for communicating ideas.' *IEEE Transactions on Software Engineering*, **SE-3**(1).

Royce, W. (1970) 'Managing the development of large software systems concepts:' WeSCON Proceedings, August 1970.

Rule, G. (1989) 'Private Correspondence' between G Rule and AL Rollo.

Rule, G. (1991) 'Experiences with FPA applied to Jackson Systems Development Projects, *European Software Cost Modelling Meeting 1991*, ESTEC, Noordwijk, Netherlands.

Rule, G. (1992) 'Technical complexity as a size driver' *European Software Cost Modelling Meeting 1992*, Munich Germany.

Rule, G. (1993) 'From measurement to estimation for highly constrained systems, *European Software Cost Modellers meeting 1993*, Bristol UK.

Sheppard, M & Turner, R. (1993), 'Real-Time function points: an industrial validation, *European Software Cost Modellers Meeting* 1993, Bristol.

Sommerville, I. (1989) *Software Engineering*. Addison Wesley Wokingham, England

Sutcliffe, A. (1988) *Jackson System Development*. Prentice-Hall International (UK), Hertfordshire.

Symons, C.R. (1988) 'Function point analysis: difficulties and improvements', *IEEE Transactions on Software Engineering*, **SE-14**(1), 2-11.

Symons, C.R. (1991) '*Software Sizing and Estimating MKII FPA*. John Wiley & Sons Ltd Chichester

Suding, (1977), '*Hobbits dwarfs and Software*', Datamation June 1977, pp 92-97.

Thayer, R.H. (1987) 'Software engineering project management a top-down view', *IEEE Software*, 1987, 15-53.

Times (1988) The Times of London December 18 1988.

UFPUG (1993), *Function points for highly constrained systems* Draft proposal of the United Kingdom Function Point User group (UFPUG) Special Interest Group on Function Points Extensions, 1993.

UFPUG, (1994), *Mark II function point analysis counting practices manual*, Issued by the UFPUG Counting Practices committee, Version 1.1 October 1994.

Van Riet, R W M (1992) 'The MERMAID Project' *European Software Cost Modelling Meeting*, Munich, 1992.

Verner, J M. Tate, G. Jackson, B. and Hayward , R J. (1989) 'Technology dependence in Function Point Analysis: A case study and critical review.' *In proceedings of the 11th International Conference on Software Engineering*. 1989, pp 375 - 382

Walston, C E & Felix, C P (1977) 'A method of programming measurement and estimation', *IBM syst . J.*, **16,**(1), pp 54-73.

Warburton, R D H (1983) 'Managing and predicting the costs of real-time software', *IEEE Transactions on Software Engineering*, **SE-9**(5), 562-569.

Whitmire , S. Galea, S. Gold, R., Basic, J., Crow, M. and Tollar, T. (1991) *"Scientific and real-time software size metrics."* Draft Technical Report, Boeing Company Jan. 1991

Wolverton, R W (1974) 'The cost of developing large-scale software', *IEEE Transactions on Computers*, June 1974, pp 615-636.

Yourdon, E. (1989) '*Modern Structured Analysis*', Prentice-Hall, Englwood Cliffs, N.J., 1989

Zave, P. (1982) 'An operational approach to requirements specification for embedded systems' *IEEE Transactions on Software Engineering*, **SE-8**(3), 250-269.

Zave, P. (1984) 'The operational versus the conventional approach to software development', *Communications of the ACM*, **27**(2), 104-118.

Zave, P. & Schell, W. (1986) 'Salient features of an executable specification language and its environment' *IEEE Transactions on Software Engineering*, **SE-12**(2), 312-325.

The Library Case Study

The Problem Statement

Due to early difficulties in obtaining real project data, this first case study is necessarily a hypothetical one, albeit representing a reasonably realistic JSD project scenario. The problem chosen is the University Library System (Cameron, 1986).

In order to create a project scenario with sufficient detail to enable the estimations to be possible, a number of assumptions - some directly implied by the problem description - have been made concerning the development and implementation environments. These assumptions are as follows:

a. The system required is considered to be of "low complexity".

b. There are no severe execution time constraints imposed upon the system.

c. The use of computer main store will not exceed 50% of the available storage

d. As the available computer systems are mature products, the stability of the underlying hardware and operating system is high.

e. The analysis team are considered nominally experienced, i.e. they have an average experience of about three years, with some exposure to JSD.

f. The programmer team is rated as being of average capability.

g. The project team overall has an experience averaging about one year on the available hardware and software systems.

h. The project team's experience of the chosen programming language, which is to be COBOL, is rated as high, i.e. greater than three years.

i. The project team will not be using software development tools other than a time-sharing operating system, a DBMS, interactive debugging tools, and interactive editor.

The Library Case Study and FPA

The Albrecht & Gaffney proposals for FPA were applied to the library case study with the results as follows:

Using the original set of correspondence rules for JSD projects (See Section 3.2), the FC estimation obtained for the library system is as given in Table A1:

Complexity Classes				
Parameter Type	*Simple*	*Average*	*Complex*	Total
IT External inputs	11 x 3	4 x 4		49
OT External outputs	5 x 4	1 x 5		25
FT Logical interface Files	22 x 7			154
EI External interface Files	1 x 5	2 x 10		25
QT External inquiries	7 x 3		1 x 6	27
FC	**Total Unadjusted Function Points**			**280**

Table A.1 Function Count Estimation

B Processing Complexity (PC) estimation

There are 14 processing complexity factors (PCFs) to be taken into account. Using the assumptions outlined earlier in respect of the library system's implementation and development environment, the weighting DI (degree of influence) that was attributed to each PCF is given in Table A2.

Processing Complexity Factor	**D**	**Processing Complexity factor**	**DI**
Data Communications	2	On-line Update	4
Distributed Functions	0	Complex Processing	1
Performance	3	Reusability	0
Heavily Used Configuration	1	Installation Ease	3
Transaction Rate	3	Operational Ease	3
On-line Data Entry	4	Multiple Sites	0
End User Efficiency	4	Facilitate Change	5
Total Degrees of Influence			**33**

Table A.2 Processing Complexity Estimation

The associated Processing Complexity Adjustment is now calculated from the Total Degree of Influence (TDI) using the following predetermined formula:

$$\begin{aligned} PCA &= 0.65 + (0.01 \times TDI) \\ &= 0.65 + (0.01 \times 33) \\ &= 0.98 \end{aligned}$$

C. Function Points Measure (FP) estimation

The Function Points Measure is calculated from the FC and PCA as follows:

FP	=	FC	x PCA
	=	280	x 0.98
	=	274.4	

D SLOC and Effort estimation

The estimates of SLOC and Effort can now be calculated from the FP measure. FPA offers a range of formulae that may be used in these calculations. The formulae that were appropriate to this hypothetical case study are those where the target language is COBOL:

The SLOC estimate S

S	=	(118.7 x FP) - 6490
	=	(118.7 x 274.4) - 6490
	=	26.08 KSLOC

Using the estimated SLOC value:

The Effort estimate W

W	=	(0.3793 x S) - 2913
	=	(0.3793 x 26,080) - 2913
	=	6979 work hours

Library case Study and COCOMO

A. Initial effort estimate

The COCOMO method was also applied to the library case study and the results are as follows:

A "small" project (less than 50 KDSI) of the kind being used in this case study (i.e. an information system) would be considered to be an organic mode project. Since the KDSI (KSLOC) figure obtained by FPA is 26.08, there is therefore little doubt that the Library System should placed in the organic mode category (especially

since, intuitively, a figure of over 50 KDSI would seem far too large anyway). In this case, the appropriate formula is:

$$MM_{nom} = 3.2(KDSI)^{1.05}$$

where MM_{nom} is the basic (nominal) effort value. Using the KSLOC figure generated by FPA (The use of FPA as a front end to COCOMO is now considered to be a dubious procedure though, it was thought worth exploring at the time this study was undertaken), this gives:

$$MM_{nom} = 3.2(26.08)^{1.05}$$
$$= 98.24 \text{ "man-months"}$$

B. Adjusting the basic effort estimate

For each cost driver:

i. Ascribe to it one of its permitted qualitative ratings;
ii Using the appropriate EM table (in this case study, the one for organic mode projects), read off the allotted EM value for that rating.

Then calculate:

$$MM_{adj} = MM_{nom} \times (EM_1 \times EM_2 \times \ldots EM_{15})$$

Table A3 gives a description of each cost driver, and a qualitative rating and corresponding EM value in respect of the Library System project; for full details of the EM tables used in COCOMO, Boehm (**ibid**.) should be consulted. Obviously, in arriving at the qualitative ratings, judgement has been based on the project scenario and associated assumptions described in an earlier section

Using the Effort Multiplier Values given in Table A3 gives:

$$MM_{adj} = MM_{nom} \times (EM_1 \times EM_2 \times \ldots EM_{15})$$
$$= 98.24 \times 0.669$$
$$= 65.68 \text{ MM}$$

Cost Drivers		Qualitative rating	EM Value
Id.	Description		
Product Attributes			
RELY	Required Software Reliability	HIGH	1.15
DATA	Relative Data Base Size	VHIGH	1.16
CPLX	Product Complexity	LOW	0.85
Computer Attributes			
TIME	Execution Time Constraint	NOM	1.00
STOR	Main Storage Constraint	NOM	1.00
VIRT	Virtual Machine Volatility	LOW	0.87
TURN	Computer Turnaround Time	LOW	0.87
Personnel Attributes			
ACAP	Analyst Capability	NOM	1.00
AEXP	Applications Experience	NOM	1.00
PCAP	Programmer Capability	NOM	1.00
VEXP	Virtual machine Experience*	NOM	1.00
LEXP	Programming Language Experience	HIGH	0.95
Project Attributes			
MODP	Modern Programming Practice	VHIGH	0.82
TOOL	Use of Software Tools	NOM	1.00
SCED	Required Software Development Schedule	NOM	1.00

* For a given software product the underlying virtual machine isthe complex of hardware and software it calls upon to accomplish its task

Table A.3 COCOMO cost drivers and effort multipliers for the library system

The "Man Month" in COCOMO is given as 152 "Man Hours", so in order to directly compare the effort figure with that obtained by FPA, which is given in work hours, the COCOMO figure can be adjusted thus:

$$W = 65.68 \times 152$$
$$= 9983.4 \text{ work hours}$$

Having calculated the overall effort, development time TDEV is obtained via the appropriate formula for an organic mode project:

$$\begin{aligned} TDEV &= 2.5 \times (MM)^{0.38} \\ &= 2.5 \times (65.68)^{0.38} \\ &= 12.26 \text{ months} \end{aligned}$$

C. Derive Effort, Development Time, and Personnel Requirements by Phase

Having obtained overall effort and development time estimations, these can now be broken down by phase using tables provided for intermediate COCOMO. For a medium size project of 32 KDSI (treating 26 KDSI as being sufficiently close), the phase breakdown figures are as given in Table 4. The final column in Table 4 gives the Full Time (Equivalent) Software Personnel (FSP) required for the various phases of the project.

FSP is calculated simply from:

$$FSP = MM/TDEV$$

Hence the overall average FSP is:

$$FSP_{avge} = 65.68/12.66$$

$$FSP_{avge} = 5.2$$

The PENSIONS Project

Applying FPA to the PENSIONS Project

The adaptations of the two FPA methods were tried out on an actual commercial JSD-based project called PENSIONS carried out in 1988 by the Abbey National Building Society for their new pensions scheme. In order to meet the launch date set by the UK government for the introduction of new pension schemes, the system had to be developed quickly once the final form of legislation was clear. Apart from the modelling phase, it took some 5 months to complete the specification and implementation of the system along with one major amendment to the JSD model caused by revisions made to the proposed legislation before it became law.

The results are presented in summary form, as full calculations were presented in Appendix A, and apart from some small changes calculations are essentially similar.

Applying Albrecht & Gaffney FPA to the PENSIONS project

Table B1 details the results obtained from analysis of PENSIONS project documentation using the counting rules from section 3.2:

	Complexity Classes			
Parameter Type	*Simple*	*Average*	*Complex*	Total
IT External inputs	56 x 3	28 x 4	7 x 6	322
OT External outputs	8 x 4	9 x 5	11 x 7	154
FT Logical interface Files	47 x 7	13 x 10	1 x 15	474
EI External interface Files	0 x 5	4 x 7	6 x 10	88
QT External inquiries	2 x 3	4 x 4	12 x 6	94
FC	**Total Unadjusted Function Points**			**1132**

Table B.1 Function Count Estimation

Table B2 lists the complexity ratings that were obtained by interviewing the PENSIONS project manager. The numbers entered in the table represent a 'degree of influence' estimate on a scale from 0-5.

Processing Complexity Factor	DI	Processing Complexity factor	DI
Data Communications	0	On-line Update	4
Distributed Functions	0	Complex Processing	1
Performance	0	Reusability	3
Heavily Used Configuration	1	Installation Ease	0
Transaction Rate	0	Operational Ease	1
On-line Data Entry	4	Multiple Sites	0
End User Efficiency	1	Facilitate Change	4
Total Degrees of Influence			19

Table B2 Processing Complexity Estimation

Processing complexity adjustment (PCA):

PCA = 0.65 + (0.01 x TDI) = 0.84

Function points measure (FP):

FP = FC x PCA = 1132 x 0.84 = 951

SLOC (S) and work effort (W) estimation:

S = (118.7) x FP) - 6490 = 106.4 KSLOC

W = (0.3793 x S) - 2913 =

Processing Complexity Factor	DI	Processing Complexity factor	DI
Data Communications	0	On-line Update	4
Distributed Functions	0	Complex Processing	1
Performance	0	Reusability	3
Heavily Used Configuration	1	Installation Ease	0
Transaction Rate	0	Operational Ease	1
On-line Data Entry	4	Multiple Sites	0
End User Efficiency	1	Facilitate Change	4
Total Degrees of Influence			19

Table B2 Processing Complexity Estimation

Processing complexity adjustment (PCA):

PCA = 0.65 + (0.01 x TDI) = 0.84

Function points measure (FP):

FP = FC x PCA = 1132 x 0.84 = 951

SLOC (S) and work effort (W) estimation:

S = (118.7) x FP) - 6490 = 106.4 KSLOC

W = (0.3793 x S) - 2913 = 37445 work hours

Applying MKII FPA to the PENSIONS Project

Bearing in mind the previous observations (section 3.4) concerning the correspondence between JSD and MKII FPA, Table B3 gives the counts obtained from the PENSIONS project data.

JSD Category	No in category	No of data element types	No of entity types
Entity actions	61	419	164
Function inputs'	41	134	32
Function outputs'	30	453	52
Inquiry functions	18	45" & 352' = 397	15

Table B3 PENSIONS transaction counts

For the purposes of using Symon's formula, this analysis produces:

Input data element types:	598
Output data element types	806
Entity reference types	263

Using Symon's weights we obtain

$$UFP = 598x\ 0.44 + 263 \times 1.67 + 806 \times 0.38 = 1009$$

Processing Complexity Factor	DI	Processing Complexity factor	DI
Applications interface	4	Documentation requirements	4
Special security	2	Special training facilities	0
Third party access	0	Client defined	0
Total Additional Degrees of Influence			10

Table B4 Symons additional technical complexity factors

Thus:

$$TDI = TDI(A\&G) + 10 = 19 + 10 = 29$$

and the following results are obtained

$$TCF = 0.65 + (0.005 \times TDI = 0.65 \times 0.145 = 0.795$$
$$FP = 1009 \times TCF = 1009 \times 0.795 = 802.155$$

applying Albrecht & Gaffney's formula to allow a comparison of work hours and SLOC

$$S = (118.7 \times FP) - 6490 = \underline{88.7\ KSLOC}$$
$$W = (0.3793 \times S) - 2913 = \underline{30,731\ Work\ Hours}$$

The two figures compare though the MKII figures are somewhat lower than the A& G figures.

Since these calculations were originally performed Symon's has revised the weights assigned to the parameters in the UFP calculations with the new weights the figures become:

$$UFP = 598x\ 0.58 + 263 \times 1.66 + 806 \times 0.26 = 993$$
$$FP = 789.4$$

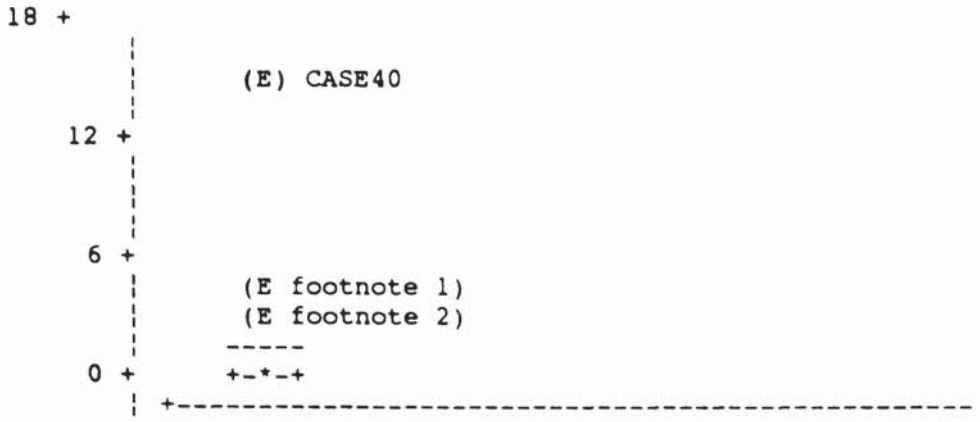
It should be noted that for the actual project the corresponding figures are some 20K work hours and 57 KSLOC.

Data and Statistics for Chapter 5

Table C1 The original Data set - outliers included

Cases	SL	IT	SQ	LV	LF	NL	BX	DSI
1	2	1	7	7	8	7	15	115
2	5	1	19	7	21	20	41	186
3	0	0	6	3	3	3	6	31
4	1	0	7	4	5	6	11	22
5	2	0	4	5	5	5	10	153
6	2	1	7	7	7	10	17	60
7	1	0	25	4	25	23	48	137
8	1	0	12	2	6	10	16	43
9	0	0	6	3	3	4	7	26
10	3	1	11	6	7	9	16	178
11	2	1	10	7	8	7	15	301
12	1	1	7	4	6	7	13	108
13	0	0	5	2	5	1	6	876
14	2	1	11	7	11	6	17	250
15	2	1	12	7	11	6	17	317
16	0	0	6	2	5	1	6	136
17	2	4	12	9	31	9	40	705
18	0	0	7	3	4	3	7	88
19	3	5	34	11	24	28	52	682
20	3	3	42	9	28	26	54	807
21	0	0	7	3	4	3	7	94
22	1	0	6	4	3	5	8	257
23	1	0	9	4	4	6	10	324
24	1	0	8	4	6	8	14	456
25	3	0	9	5	9	13	22	678
26	3	0	9	5	9	13	22	661
27	3	0	9	5	9	13	22	666
28	1	0	3	4	3	5	8	274
29	1	0	4	4	4	6	10	344
30	1	0	4	4	4	6	10	314
31	2	0	7	5	7	10	17	518
32	3	0	14	5	14	18	32	1028
33	1	0	5	4	5	7	12	392
34	1	0	3	4	3	5	8	264
35	3	0	11	5	10	15	25	830
36	1	0	3	4	3	5	8	301
37	1	0	3	4	3	5	8	272
38	5	1	17	8	23	19	42	689
39	1	4	31	14	26	26	52	362
40	9	16	37	9	57	39	96	563
41	4	1	16	6	11	13	24	216
42	21	4	59	13	55	61	116	644
43	1	2	10	6	8	13	21	131
44	2	1	12	5	11	15	26	298
45	1	3	23	11	16	19	35	488
46	0	1	3	7	8	5	13	221
47	3	5	15	10	19	24	43	654
48	0	1	6	7	17	10	27	188
49	2	1	4	7	8	6	14	426
50	1	2	17	15	18	20	38	524
51	10	2	17	9	29	19	48	520

```
EXAMINE /VARIABLES SL IT SQ LV LF NL BX DSI.

SL
Valid cases: 51.0   Missing cases: .0   Percent missing: .0
 Mean    2.3333  Std Err   .4641  Min     .0000  Skewness 4.0881
 Median 1.0000  Variance10.9867  Max  21.000   S E Skew  .3335
 5%Trim 1.8094  Std Dev  3.3146  Range21.0000  Kurtosis20.5844
 IQR     2.0000  S E Kurt  .6559
Frequency     Stem &  Leaf

      8.00    0      .  00000000
     18.00    1      .  000000000000000000
     10.00    2      .  0000000000
      9.00    3      .  000000000
      1.00    4      .  0
      2.00    5      .  00
      3.00 Extremes    (9.0), (10.0), (21.0)
 Stem width:          1
 Each leaf:     1 case(s)
          24 +
             |
             |       (E) CASE42
             |
          16 +
             |
             |
             |       (E) CASE51
           8 +       (E) CASE40
             |
             |      -----
             |      +---+
           0 +      +-*-+
             |  +-----------------------------------------------

N of Cases          51.00
Symbol Key:  * - Median (O)  - Outlier    (E)  - Extreme

IT

Valid cases:  51.0  Missing cases: .0   Percent missing: .0


 Mean    1.2549 Std Err   .3530  Min     .0000  Skewness  4.3324
 Median 1.0000 Variance 6.3537  Max  16.0000  S E Skew   .3335
 5% Trim .8769 Std Dev  2.5207 Range 16.0000  Kurtosis 23.5789
 IQR     1.0000 S E Kurt  .6559
Frequency     Stem &  Leaf

     26.00    0  *  00000000000000000000000000
        .00   0  .
     14.00    1  *  00000000000000
        .00   1  .
      3.00    2  *  000
      8.00 Extremes    (3.0), (4.0), (5.0), (16.0)

 Stem width:       1
 Each leaf:     1 case(s)
```



Variables IT

N of Cases 51.00

Symbol Key: * - Median (O) - Outlier (E) - Extreme
Boxplot footnotes denote the following:

1) CASE19, CASE47

2) CASE17, CASE20, CASE39, CASE42, CASE45

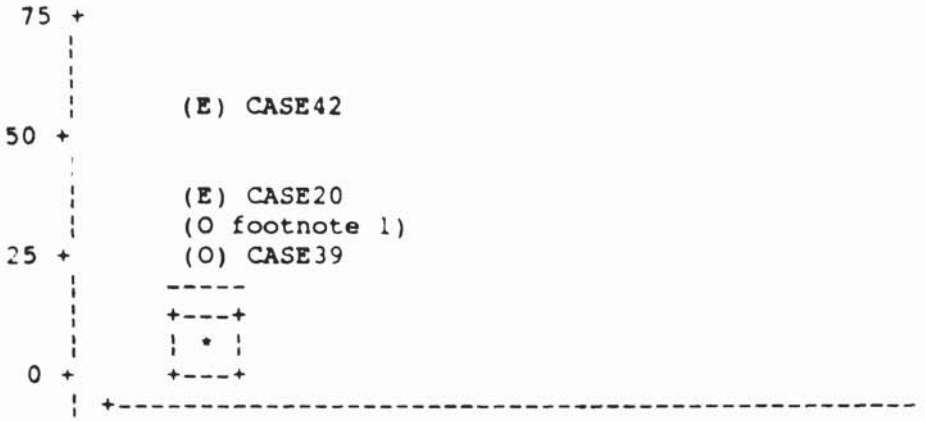
SQ

Valid cases: 51.0 Missing cases: .0 Percent missing: .0

Mean	12.3725	Std Err	1.5583	Min 3.0000	Skewness 2.3088
Median	9.0000	Variance	123.8384	Max 59.0000	S E Skew .3335
5% Trim	10.9368	Std Dev	11.1283	Range 56.0000	Kurtosis 6.1123
IQR	9.0000	S E Kurt	.6559		

Frequency	Stem &	Leaf
9.00	0 *	333334444
19.00	0 .	5566666777777789999
10.00	1 *	0011122224
6.00	1 .	567779
1.00	2 *	3
1.00	2 .	5
5.00 Extremes		(31), (34), (37), (42), (59)

Stem width: 10
Each leaf: 1 case(s)



N of Cases 51.00
Symbol Key: * - Median (O) - Outlier (E) - Extreme

Boxplot footnotes denote the following:

1) CASE19, CASE40

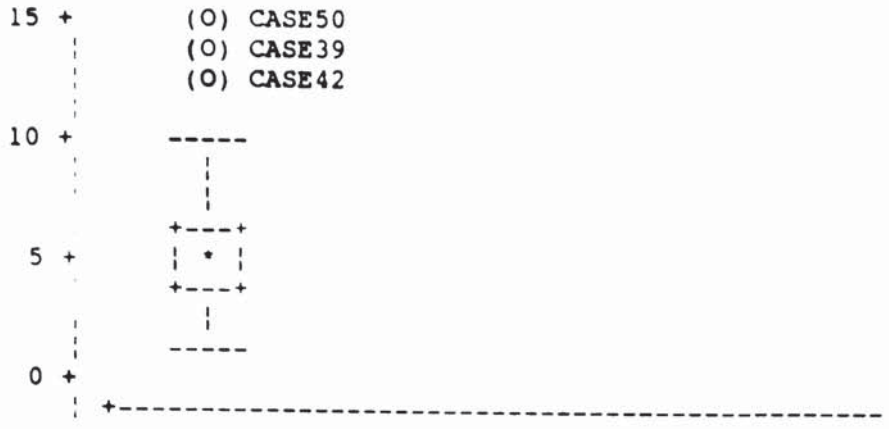
LV

Valid cases: 51.0 Missing cases: .0 Percent missing: .0

Mean	6.0588	Std Err	.4214	Min	2.0000	Skewness	1.1694
Median	5.0000	Variance	9.0565	Max	15.0000	S E Skew	.3335
5% Trim	5.8333	Std Dev	3.0094	Range	13.0000	Kurtosis	1.1695
IQR	3.0000	S E Kurt	.6559				

Frequency	Stem &	Leaf
3.00	2	. 000
4.00	3	. 0000
13.00	4	. 0000000000000
8.00	5	. 00000000
3.00	6	. 000
9.00	7	. 000000000
1.00	8	. 0
4.00	9	. 0000
1.00	10	. 0
2.00	11	. 00

3.00 Extremes (13.0), (14.0), (15.0)
Stem width: 1
Each leaf: 1 case(s)



N of Cases 51.00
Symbol Key: * - Median (O) - Outlier (E) - Extreme

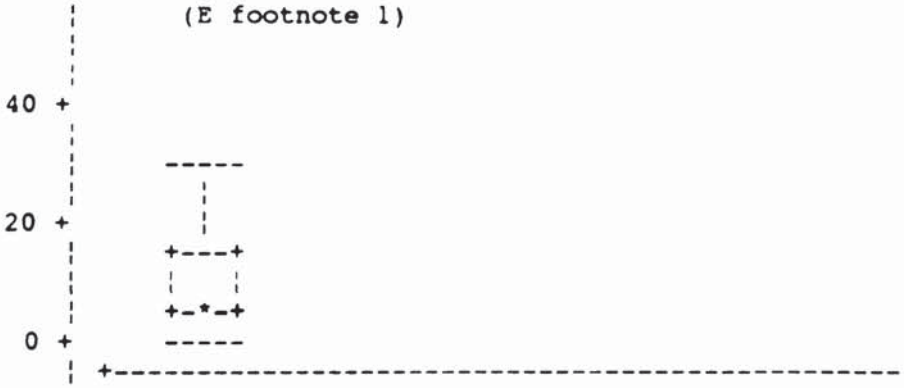
LF

Valid cases: 51.0 **Missing cases:** .0 **Percent missing:** .0

Mean	12.3333	**Std Err** 1.6622	**Min** 3.0000 **Skewness** 2.2168
Median	8.0000	**Variance** 140.9067	**Max** 57.0000 **S E Skew** .3335
5% Trim	10.7255	**Std Dev** 11.8704	**Range** 54.0000 **Kurtosis** 5.6203
IQR	12.0000	**S E Kurt** .6559	

Frequency	**Stem** &	**Leaf**
12.00	0 *	333333344444
19.00	0 .	5555566677788888999
6.00	1 *	011114
4.00	1 .	6789
3.00	2 *	134
4.00	2 .	5689
1.00	3 *	1
2.00 **Extremes**		(55), (57)

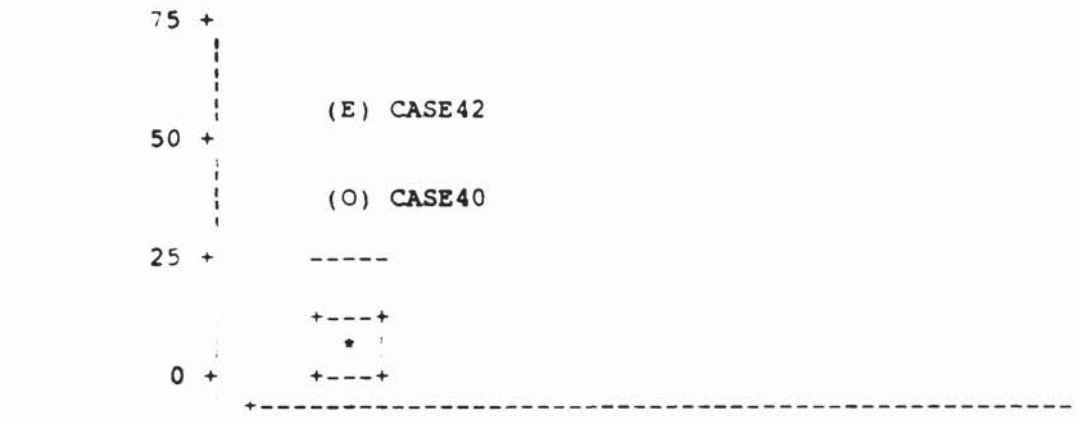
Stem width: 10
Each leaf: 1 case(s)



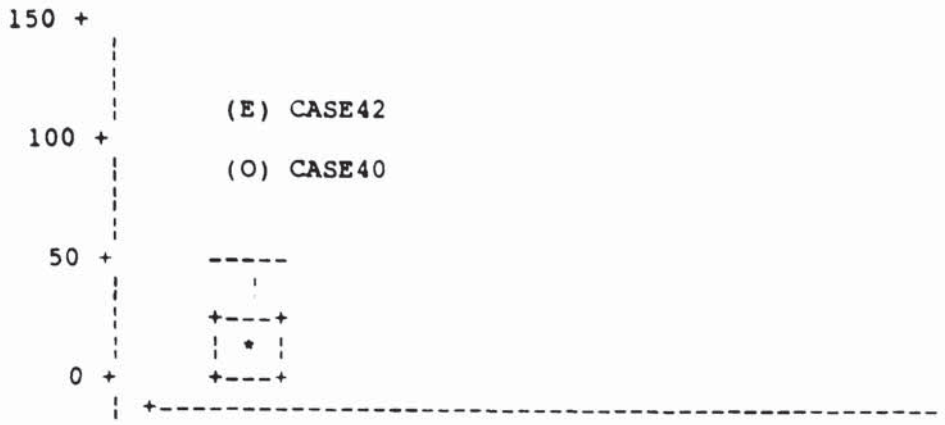
N of Cases 51.00
Symbol Key: * - Median (O) - Outlier (E) - Extreme
Boxplot footnotes denote the following:

1) CASE40, CASE42

```
NL

Valid cases:  51.0 Missing cases: .0 Percent missing: .0

Mean     12.2157 Std Err   1.4910  Min    1.0000  Skewness  2.3919
Median  9.0000  Variance113.3725  Max  61.0000  S E Skew   .3335
5% Triml0.9793  Std Dev  10.6477  Range60.0000  Kurtosis  8.1303
IQR    13.0000  S E Kurt   .6559
Frequency     Stem &  Leaf

      6.00    0  *  113334
     21.00    0  .  555555566666667777899
      9.00    1  *  000033333
      6.00    1  .  558999
      4.00    2  *  0034
      3.00    2  .  668
      2.00 Extremes    (39), (61)
 Stem width:         10
 Each leaf:     1 case(s)
```




```
N of Cases          51.00
Symbol Key: * - Median (O)  - Outlier    (E)  - Extreme


BX

Valid cases:  51.0 Missing cases: .0 Percent missing: .0


 Mean   24.5490  Std Err    3.0706 Min    6.0000  Skewness 2.2921
 Median 17.0000 Variance 480.8525 Max  116.0000  S E Skew  .3335
 5% Trim21.6776 Std Dev   21.9284 Rangel10.0000  Kurtosis 6.5878
 IQR    25.0000 S E Kurt    .6559

Frequency     Stem &  Leaf

     11.00    0      .  66677788888
     18.00    1      .  000012334455667777
      8.00    2      .  12224567
      3.00    3      .  258
      6.00    4      .  012388
      3.00    5      .  224
      2.00 Extremes    (96), (116)
 Stem width:         10
 Each leaf:     1 case(s)
```



N of Cases 51.00
Symbol Key: * - Median (O) - Outlier (E) - Extreme

DSI

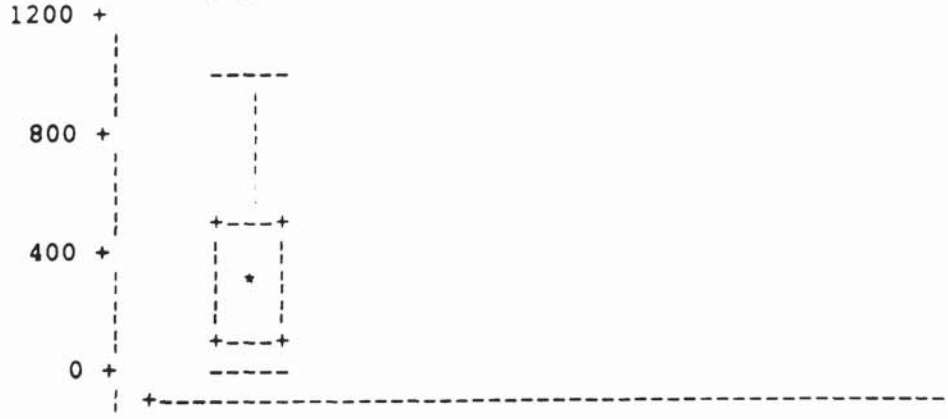
Valid cases: 51.0 Missing cases: .0 Percent missing: .0

Mean	368.9804	Std Err	35.6814	Min	22.000	Skewness	.6302
Median	301.0000	Variance	64931.10	Max	1028.000	S E Skew	.3335
5% Trim	357.1340	Std Dev	254.8158	Range	1006.000	Kurtosis	-.4697
IQR	410.0000	S E Kurt	.6559				

DSI

Frequency		Stem &	Leaf
16.00	0	*	0000000111111111
16.00	0	t	2222222233333333
7.00	0	f	4445555
8.00	0	s	66666667
3.00	0	.	888
1.00	1	*	0

Stem width: 1000
Each leaf: 1 case(s)



N of Cases 51.00
Symbol Key: * - Median (O) - Outlier (E) - Extreme

```
REGRESSION /VARIABLES SL IT SQ LV LF NL BX DSI
/DESCRIPTIVES /ORIGIN /DEPENDENT DSI /METHOD ENTER.

* * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *

Correlation:

          SL      IT      SQ      LV      LF      NL      BX
SL     1.000    .589    .823    .691    .849    .872    .871
IT      .589   1.000    .707    .629    .813    .716    .776
SQ      .823    .707   1.000    .870    .942    .971    .968
LV      .691    .629    .870   1.000    .862    .882    .882
LF      .849    .813    .942    .862   1.000    .952    .989
NL      .872    .716    .971    .882    .952   1.000    .987
BX      .871    .776    .968    .882    .989    .987   1.000
DSI     .641    .508    .762    .832    .764    .790    .786
```

The outliers were removed and the following figures obtained

```
REGRESSION /VARIABLES SL IT SQ LV LF NL BX DSI

/DESCRIPTIVES DEFAULT

 /ORIGIN /DEPENDENT DSI /METHOD ENTER /RESIDUALS

DEFAULTS OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *

Listwise Deletion of Missing Data

Coefficients have been calculated through the Origin.
The observed mean is printed.

              Mean  Std Dev  Label

SL           1.854    2.521
IT            .917    1.620
SQ          11.042   13.787
LV           5.938    6.569
LF          10.667   13.310
NL          10.875   12.939
BX          21.542   25.848
DSI        348.646  425.513

N of Cases =     48
```

```
Correlation:
          SL       IT       SQ       LV       LF       NL       BX

SL     1.000     .561     .736     .760     .809     .804     .819
IT      .561    1.000     .785     .801     .813     .774     .806
SQ      .736     .785    1.000     .886     .929     .956     .957
LV      .760     .801     .886    1.000     .910     .920     .929
LF      .809     .813     .929     .910    1.000     .939     .985
NL      .804     .774     .956     .920     .939    1.000     .984
BX      .819     .806     .957     .929     .985     .984    1.000
DSI     .783     .627     .797     .841     .827     .864     .859

Beginning Block Number  1.  Method:  Enter

REGRESSION  /VARIABLES  SL  IT  SQ  DSI/DESCRIPTIVES  NONE
/ORIGIN

/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

* * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * *
                          *

Listwise Deletion of Missing Data

Equation Number 1    Dependent Variable..   DSI

Beginning Block Number  1.  Method:  Enter


Variable(s) Entered on Step Number
   1..     SQ
   2..     SL
   3..     IT


Multiple R            .84813
R Square              .71932
Adjusted R Square     .70061
Standard Error     232.82586


Analysis of Variance
                 DF       Sum of Squares       Mean Square
Regression       3                             6251592.39690
2083864.13230
Residual         45       2439354.60310
54207.88007
F =              38.44209                Signif F =  .0000
----------------- Variables in the Equation ------------
```

```
Variable          B          SE B       Beta        T    Sig T
SQ          14.38849       4.81866     .46620    2.986 .0046
SL          72.23471      19.71300     .42792    3.664 .0007
IT           5.35775      33.52958     .02040     .160 .8738
End Block Number   1   All requested variables entered.
Total Cases =      48
Outliers - Standardized Residual

  Case #        *ZRESID
      31         2.61937
      48        -1.96570
      34         1.95435
       2        -1.94958
      16         1.57387


REGRESSION /VARIABLES BX DSI /DESCRIPTIVES NONE /ORIGIN

/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *

Listwise Deletion of Missing Data

Equation Number 1    Dependent Variable..   DSI

Beginning Block Number  1.  Method:  Enter

Variable(s) Entered on Step Number
   1..    BX


Multiple R            .85874
R Square              .73743
Adjusted R Square     .73184
Standard Error     220.34871

Analysis of Variance
                DF        Sum of Squares       Mean Square
Regression       1        6408929.93081        6408929.93081
Residual        47        2282017.06919          48553.55466
F =          131.99713                  Signif F =  .0000
------------------ Variables in the Equation ------------
Variable          B          SE B       Beta        T     Sig T

BX          14.13655       1.23044     .85874    11.489  .0000


End Block Number   1   All requested variables entered
```

Outliers - Standardized Residual

Case #	*ZRESID
31	2.61236
7	-2.45771
34	2.16287
2	-1.78625
38	-1.69323

```
REGRESSION /VARIABLES NL LF DSI /DESCRIPTIVES NONE /ORIGIN
/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *

Listwise Deletion of Missing Data

Equation Number 1    Dependent Variable..   DSI

Beginning Block Number  1.  Method:  Enter

Variable(s) Entered on Step Number
   1..     LF
   2..     NL


Multiple R            .86559
R Square              .74925
Adjusted R Square     .73835
Standard Error     217.65856

Analysis of Variance
                    DF       Sum of Squares       Mean Square
Regression           2       6511685.59128  3255842.79564
Residual            46       2179261.40872    47375.24802
F =                 68.72455          Signif F =  .0000
 ------------------ Variables in the Equation -------------

Variable     B          SE B       Beta          T    Sig T

LF         4.16342     6.88001     .13024       .605   .5481
NL        24.40505     7.07751     .74211      3.448   .0012
End Block Number   1   All requested variables entered.
```

```
Outliers - Standardized Residual
  Case #        *ZRESID
      31         2.43694
       7        -2.42766
      34         1.94015
       2        -1.78965
      38        -1.74944

REGRESSION /VARIABLES LV DSI /DESCRIPTIVES NONE /ORIGIN
/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

  * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *


Listwise Deletion of Missing Data

Equation Number 1     Dependent Variable..    DSI
Beginning Block Number  1.  Method:  Enter

Variable(s) Entered on Step Number
   1..    LV

Multiple R              .84102
R Square                .70731
Adjusted R Square       .70108
Standard Error       232.64232

Analysis of Variance
                     DF        Sum of Squares      Mean
Square Regression     1         6147191.96572
6147191.96572
Residual             47         2543755.03428     54122.44754
F =                 113.57934              Signif F =  .0000
------------------ Variables in the Equation ------------

Variable            B          SE B       Beta        T       Sig T
LV              54.48141      5.11209    .84102     10.657    .0000

Outliers - Standardized Residual

  Case #        *ZRESID
      31         3.24787
      34         2.39678
      24         1.74342
      38        -1.72256
```

```
REGRESSION /VARIABLES LV LF NL DSI /DESCRIPTIVES NONE
/ORIGIN
/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *


Listwise Deletion of Missing Data

Equation Number 1     Dependent Variable..    DSI


Beginning Block Number  1.  Method:  Enter
Variable(s) Entered on Step Number
   1..     NL
   2..     LV
   3..     LF


Multiple R              .87221
R Square                .76075
Adjusted R Square       .74479
Standard Error       214.96003

Analysis of Variance

                   DF        Sum of Squares         Mean Square
Regression          3                            6611595.38905
2203865.12968
Residual           45                            2079351.61095
46207.81358
F =                47.6946                 Signif F =  .0000

------------------ Variables in the Equation ------------

Variable         B          SE B        Beta        T     Sig T

NL            19.05575   7.87979      .57945    2.418   .0197
LV            18.88687  12.84440      .29155    1.470   .1484
LF            .56610     7.22171      .01771     .078   .9379

End Block Number   1   All requested variables entered.

Outliers - Standardized Residual
  Case #          *ZRESID
      31           2.71044
      34           2.06582
      38          -1.91935
       7          -1.81886
      16           1.60945
```

```
REGRESSION  /VARIABLES  LV  BX  DSI  /DESCRIPTIVES  NONE
/ORIGIN
/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *

Listwise Deletion of Missing Data

Equation Number 1     Dependent Variable..    DSI

Beginning Block Number  1.  Method:  EnterDSI


Variable(s) Entered on Step Number
   1..     BX
   2..     LV


Multiple R            .86658
R Square              .75096
Adjusted R Square     .74013
Standard Error     216.91683

Analysis of Variance

                    DF        Sum of Squares        Mean
Square
Regression           2        6526513.17916
3263256.58958
Residual            46         2164433.82084
47052.90915
F =                 69.35292          Signif F =  .0000
------------------ Variables in the Equation ------------

Variable         B          SE B        Beta        T     Sig T
BX            9.31609     3.28113     .56591     2.839   .0067
LV           20.41090    12.91168     .31508     1.581   .1208


End Block Number   1   All requested variables entered.


Outliers - Standardized Residual

  Case #        *ZRESID
      31         2.89434
      34         2.28218
      38        -1.88178
       7        -1.80629
      24         1.71029
```

```
REGRESSION /VARIABLES LV SL IT SQ DSI /DESCRIPTIVES NONE
/ORIGIN
/DEPENDENT DSI /METHOD ENTER /RESIDUALS DEFAULT OUTLIERS.

 * * * *  MULTIPLE REGRESSION THROUGH THE ORIGIN  * * * *


Listwise Deletion of Missing Data

Equation Number 1    Dependent Variable..   DSI
Beginning Block Number  1.  Method:  Enter
Variable(s) Entered on Step Number
   1..     SQ
   2..     SL
   3..     IT
   4..     LV

Multiple R            .87539
R Square              .76631
Adjusted R Square     .74506
Standard Error     214.84714

Analysis of Variance

                 DF       Sum of Squares       Mean Square
Regression        4        6659938.06448     1664984.51612
Residual         44        2031008.93552       46159.29399
F =              36.07041              Signif F =  .0000

------------------ Variables in the Equation ------------

Variable     B          SE B          Beta          T       Sig T

SQ         6.26561     5.21828       .20301       1.201     .2363
SL        49.69297    19.70642       .29438       2.522     .0154
IT       -35.23465    33.81673      -.13416      -1.042     .3031
LV        35.29020    11.86505       .54477       2.974     .0048


End Block Number   1   All requested variables entered.

Outliers - Standardized Residual

  Case #        *ZRESID
      31         2.86134
      34         2.02725
       2        -1.83064
      16         1.64654
      48        -1.53871
```

Using the original data forecasts were made using each model

the resulting data and statisitics are given below

Table C2 Forecasts made Using original data

M1 forecast	M2 forecast	M3 forecast	M4 forecast	M5 forecast	M6 forecast	M7 forecast
250.6	210	202.8	381.5	270.1	282.3	355.4
640	574	572	381.5	526.2	524.1	580.1
86.4	84	85.2	163.5	115.5	117	143.7
173	154	166.4	218	192.8	183.9	235
202	140	142	272.5	192.5	195	301.1
250.6	238	272	381.5	326.8	300.9	355.4
432.2	672	661.2	218	529.5	528	348.4
245	224	268	109	232.2	189.6	195.9
86.4	98	109.6	163.5	134.6	126.3	143.7
380.4	224	247.6	327	288.9	271.2	395
293.8	210	202.8	381.5	270.1	282.3	374.3
178.4	182	194.8	218	212.5	202.5	199.8
308.2	238	190.4	381.5	252.8	300.9	380.6
322.6	238	190.4	381.5	252.8	300.9	386.9
86.4	84	44.4	109	59.7	96.6	108.4
338.8	560	343.6	490.5	359.7	555.6	351.9
100.8	98	89.2	163.5	116.1	126.3	150
733.2	728	779.2	599.5	756	708	575.6
837.6	756	746.4	490.5	682.6	685.8	625.8
100.8	98	89.2	163.5	116.1	126.3	150
158.6	112	134	218	172.5	156	228.7
202.8	140	162.4	218	192.2	174.6	247.6
187.4	196	219.2	218	231.6	211.8	241.3
346.2	308	353.2	272.5	347.7	306.6	382.3
346.2	308	353.2	272.5	347.7	306.6	382.3
346.2	308	353.2	272.5	347.7	306.6	382.3
115.4	112	134	218	172.5	156	209.8
129.8	140	162.4	218	192.2	174.6	216.1
129.8	140	162.4	218	192.2	174.6	216.1
245.2	238	272	272.5	289.2	260.1	320
418.2	448	495.2	272.5	446.2	399.6	413.8
144.2	168	190.8	218	211.9	193.2	222.4
115.4	112	134	218	172.5	156	209.8
375	350	406	272.5	386.5	334.5	394.9
115.4	112	134	218	172.5	156	209.8
115.4	112	134	218	172.5	156	209.8
611.2	588	555.6	436	527.1	553.8	602.8
540.2	728	738.4	763	775.4	769.2	598.4
524.6	336	361.2	327	367.7	345.6	476.2
227	294	349.2	327	365.9	317.7	254.1
322.6	364	410	272.5	387.1	343.8	316.3
419.6	490	527.6	599.5	579.3	549.9	477.3
48.6	182	154	381.5	231.9	263.7	230.8
459.6	602	661.6	545	657.8	603.9	420.6
91.8	378	312	381.5	332.8	393.9	249.7
207.4	196	178.4	381.5	251	273	336.5
327.8	532	560	817.5	674.8	659.4	615.9
977.6	672	579.6	490.5	549.5	630	851.4

```
COMPUTE M1fcast=(14.4 * SQ)+(72.2 * SL)+(5.4*IT).
COMPUTE M2fcast=(14*BX).
COMPUTE M3fcast=(4*LF)+(24.4*NL).
COMPUTE M4fcast=(54.5* LV).
COMPUTE M5fcast=(19.1*NL)+(18.8* LV)+(0.6*LF).
COMPUTE M6fcast=(9.3*BX)+(20.4* LV).
COMPUTE M7fcast=(6.3*SQ)+(49.7*SL)+(-35.2*IT)+(35.3* LV).

The raw data or transformation pass is proceeding

     48 cases are written to the uncompressed active

file.
Number of cases read =  48    Number of cases listed = 48
T-TEST /PAIRS M1FCAST M2FCAST M3FCAST M4FCAST M5FCAST
M6FCAST M7FCAST WITH DSI.


Paired samples t-test:  M1FCAST
                        DSI

Variable    Number                  Standard   Standard
           of Cases      Mean      Deviation     Error

M1FCAST        48     297.8208      206.194     29.762
DSI            48     348.6458      246.523     35.583

(Diff)Standard Standard| 2-Tail     | t  Degrees of 2-Tail
Mean  Deviation   Error|Corr. Prob.|Value Freedom  Prob.
                       |           |
-50.825 221.953  32.036|.531 .000  |-1.59   47       .119
-------------------------------------------------------
Paired samples t-test:  M2FCAST
                        DSI

Variable    Number                  Standard   Standard
           of Cases      Mean      Deviation     Error
M2FCAST        48     301.5833      202.116     29.173
DSI            48     348.6458      246.523     35.583


(Diff) Standard Standard|    2-Tail| t  Degrees of 2-Tail
 Mean  Deviation  Error |Corr. Prob|Value Freedom   Prob.
                        |          |
 -47.063 215.184  31.059|.555 .000 |-1.52    47     .136
```

Paired samples t-test: M3FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M3FCAST	48	308.0167	200.114	28.884
DSI	48	348.6458	246.523	35.583

(Dif) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
-40.63	211.392	30.51	.569	.000	-1.33	47	.189

Paired samples t-test: M4FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M4FCAST	48	323.5938	154.725	22.333
DSI	48	348.6458	246.523	35.583

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
-25.05	231.261	.380	.409	.004	-.75	47	.457

Paired samples t-test: M5FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M5FCAST	48	325.7375	180.686	26.080
DSI	48	348.6458	246.523	35.583

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
-22.91	209.059	30.175	.558	.000	-.76	47	.452

Paired samples t-test: M6FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M6FCAST	48	321.4625	181.556	26.205
DSI	48	348.6458	246.523	35.583

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
-27.18	212.83	30.720	.541	.000	-.88	47	.381

Paired samples t-test: M7FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M7FCAST	48	339.0417	157.240	22.696
DSI	48	348.6458	246.523	35.583

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
-9.60	207.651	29.972	.547	.000	-.32	47	.750

FINISH.

The second data set was then used to produce a set of forecasts the resulting data and statistics are reproduced below

Table C3 The Second Data Set

Cases	SL	IT	SQ	LV	LF	NL	BX	DSI
1	148	2	102	8	63	188	251	1332
2	6	0	17	6	14	9	23	152
3	26	0	26	4	6	27	33	136
4	4	0	11	5	8	7	15	139
5	25	1	13	5	10	29	39	149
6	8	1	9	5	6	12	18	134
7	19	2	13	5	10	24	34	143
8	2	2	13	7	8	9	17	140
9	2	1	12	5	8	7	15	138
10	0	1	9	5	6	4	10	135
11	0	0	8	3	6	2	8	137
12	0	2	36	7	13	26	39	189
13	0	2	36	7	26	13	39	197
14	2	0	5	4	6	3	9	135
15	6	1	16	5	14	9	23	154
16	5	1	23	5	21	8	29	173
17	2	2	10	5	8	6	14	142
18	112	1	144	8	108	149	257	798
19	9	1	19	7	14	15	29	315
20	7	0	20	8	14	13	27	305
21	66	0	71	4	67	70	137	403
22	51	0	79	9	63	68	131	573
23	0	0	8	3	6	2	8	137
24	20	1	42	10	32	31	63	413

Forecasts were as follows:

Table C4 Forecasts made using data set two

M1FCAST	M2FCAST	M3FCAST	M4FCAST	M5FCAST	M6FCAST	M7FCAST
12165.2	3514	4839.2	436	3779	2497.5	8210.2
678	322	275.6	327	293.1	336.3	617.1
2251.6	462	682.8	218	594.5	388.5	1597.2
447.2	210	202.8	272.5	232.5	241.5	444.6
1997.6	546	747.6	272.5	653.9	464.7	1465.7
712.6	252	316.8	272.5	326.8	269.4	595.6
1569.8	476	625.6	272.5	558.4	418.2	1132.3
342.4	238	251.6	381.5	308.3	300.9	358
322.6	210	202.8	272.5	232.5	241.5	316.3
135	140	121.6	272.5	174	195	198
115.2	112	72.8	163.5	98.2	135.6	156.3
529.2	546	686.4	381.5	636	505.5	403.5
529.2	546	421.2	381.5	395.5	505.5	403.5
216.4	126	97.2	218	136.1	165.3	272.1
669	322	275.6	272.5	274.3	315.9	540.3
697.6	406	279.2	272.5	259.4	371.7	534.7
299.2	196	178.4	272.5	213.4	232.2	268.5
10165.4	3598	4067.6	436	3061.1	2553.3	6720.8
928.8	406	422	381.5	426.5	412.5	778.9
793.4	378	373.2	436	407.1	414.3	756.3
5787.6	1918	1976	218	1452.4	1355.7	3868.7
4819.8	1834	1911.2	490.5	1505.8	1401.9	3350.1
115.2	112	72.8	163.5	98.2	135.6	156.3
2054.2	882	884.4	545	799.3	789.9	1576.4

```
COMPUTE M1fcast=(14.4 * SQ)+(72.2 * SL)+(5.4*IT).
COMPUTE M2fcast=(14*BX).
COMPUTE M3fcast=(4*LF)+(24.4*NL).
COMPUTE M4fcast=(54.5* LV).
COMPUTE M5fcast=(19.1*NL)+(18.8* LV)+(0.6*LF).
COMPUTE M6fcast=(9.3*BX)+(20.4* LV).
COMPUTE      M7fcast=(6.3*SQ)+(49.7*SL)+(-35.2*IT)+(35.3*
LV).
The raw data or transformation pass is proceeding
     24  cases  are  written  to  the  uncompressed  active
file.
T-TEST /PAIRS  M1FCAST  M2FCAST  M3FCAST  M4FCAST  M5FCAST
M6FCAST
M7FCAST WITH DSI.
Paired samples t-test:  M1FCAST  DSI

Variable        Number                    Standard    Standard
                of Cases      Mean        Deviation     Error
M1FCAST           24       2014.2583     3169.015     646.872
DSI               24        277.8750      279.280      57.008
```

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	Prob	t Value	Degrees of Freedom	2-Tail Prob.
1736.4	2908.705	593.737	.938	.000	2.92	23	.008

Paired samples t-test: M2FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M2FCAST	24	739.6667	984.777	201.017
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	Prob	t Value	Degrees of Freedom	2-Tail Prob.
461.8	733.528	149.731	.927	.000	3.08	23	.005

Paired samples t-test: M3FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M3FCAST	24	832.6833	1226.285	250.314
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	Prob	t Value	Degrees of Freedom	2-Tail Prob.
554.8	964.618	196.902	.951	.000	2.82	23	.010

Paired samples t-test: M4FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M4FCAST	24	317.9167	101.219	20.661
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	Prob	t Value	Degrees of Freedom	2-Tail Prob.
40.04	237.512	48.482	.563	.004	.83	23	.417

Paired samples t-test: M5FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M5FCAST	24	704.8458	918.339	187.455
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	Prob	t Value	Degrees of Freedom	2-Tail Prob.
426.97	655.715	133.847	.958	.000	3.19	23	.004

Paired samples t-test: M6FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M6FCAST	24	610.3500	673.242	137.425
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
332.48	425.13	86.780	.932	.000	3.83	23	.001

Paired samples t-test: M7FCAST
DSI

Variable	Number of Cases	Mean	Standard Deviation	Standard Error
M7FCAST	24	1446.7250	2090.126	426.645
DSI	24	277.8750	279.280	57.008

(Diff) Mean	Standard Deviation	Standard Error	2-Tail Corr.	2-Tail Prob	t Value	Degrees of Freedom	2-Tail Prob.
1168.9	1829.16	373.38	.943	.000	3.13	23	.005

JSD-COCOMO Phase Sensitive Cost Driver EM Values

Table D1 Process Level Effort Multipliers

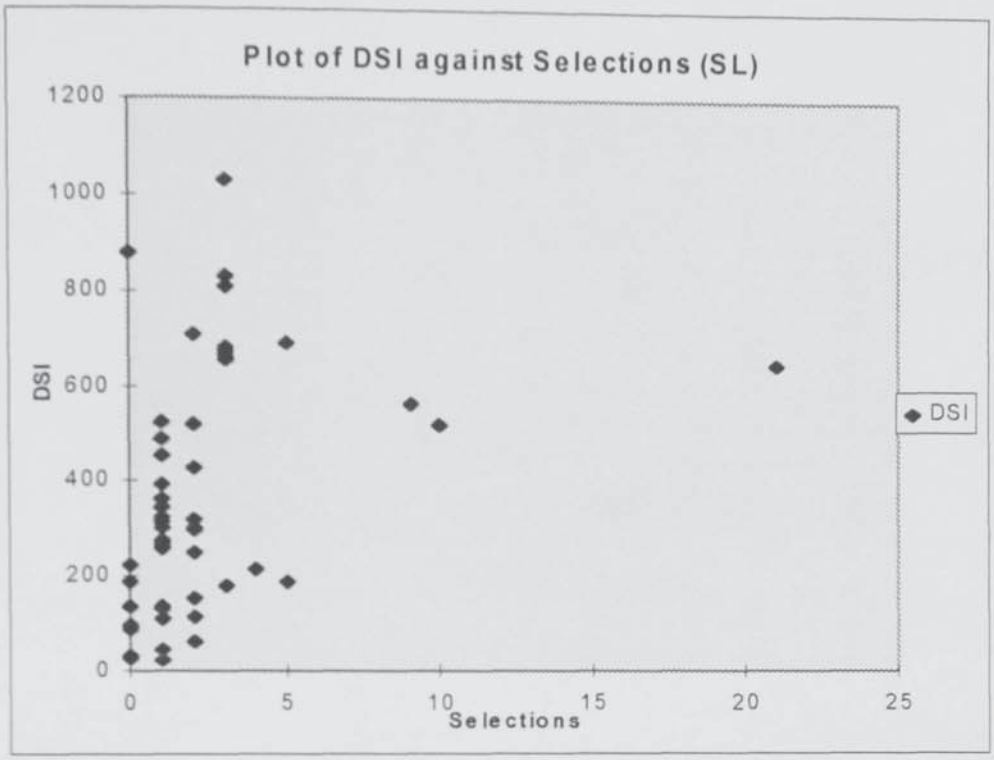
	Rating	AR	NS	PD	CO	IN
CPLX	v low	0.70	0.70	0.70	0.70	0.70
	low	0.85	0.85	0.85	0.85	0.85
	nominal	1.0	1.0	1.0	1.0	1.0
	high	1.15	1.15	1.15	1.15	1.15
	v high	1.30	1.30	1.30	1.30	1.30
	extra high	1.65	1.65	1.65	1.65	1.65
PCAP	v low	1.00	1.00	1.50	1.50	1.50
	low	1.00	1.00	1.20	1.20	1.20
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.00	1.00	0.83	0.83	0.83
	v high	1.00	1.00	0.65	0.65	0.65
VEXP	v low	1.10	1.10	1.10	1.30	1.30
	low	1.05	1.05	1.05	1.15	1.15
	nominal	1.00	1.00	1.00	1.00	1.00
	high	0.90	0.90	0.99	0.90	0.90
LEXP	v low	1.00	1.02	1.10	1.20	1.20
	low	1.00	1.00	1.05	1.10	1.10
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.00	1.00	0.98	0.92	0.92

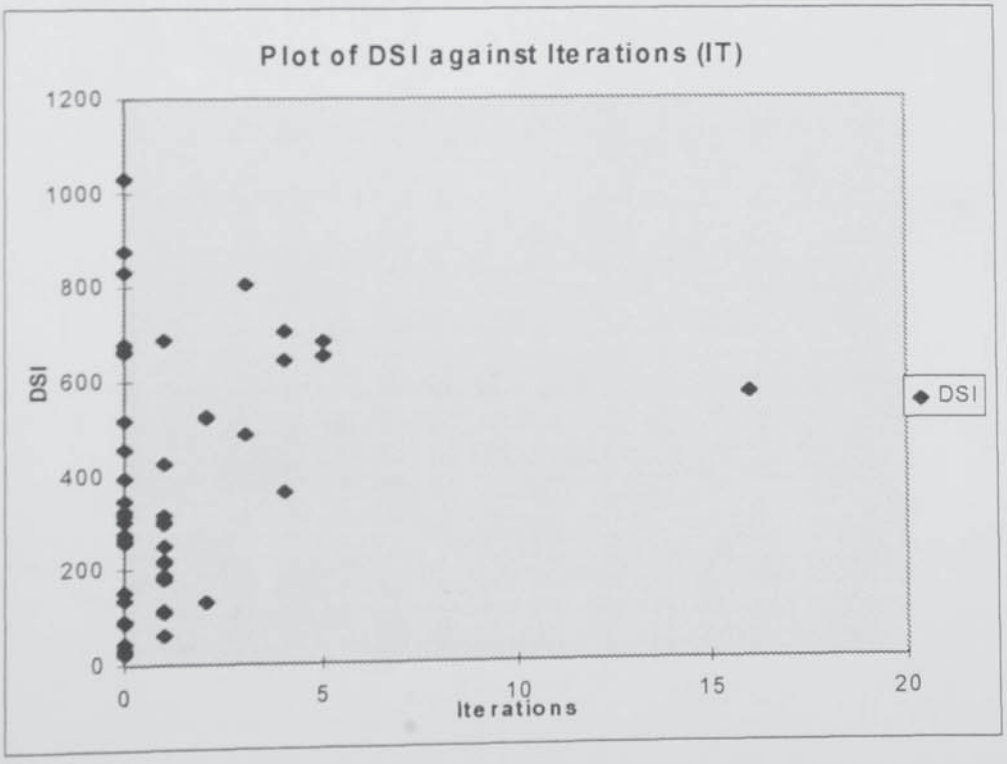
Table D2 Sub-Network Level Effort Multipliers

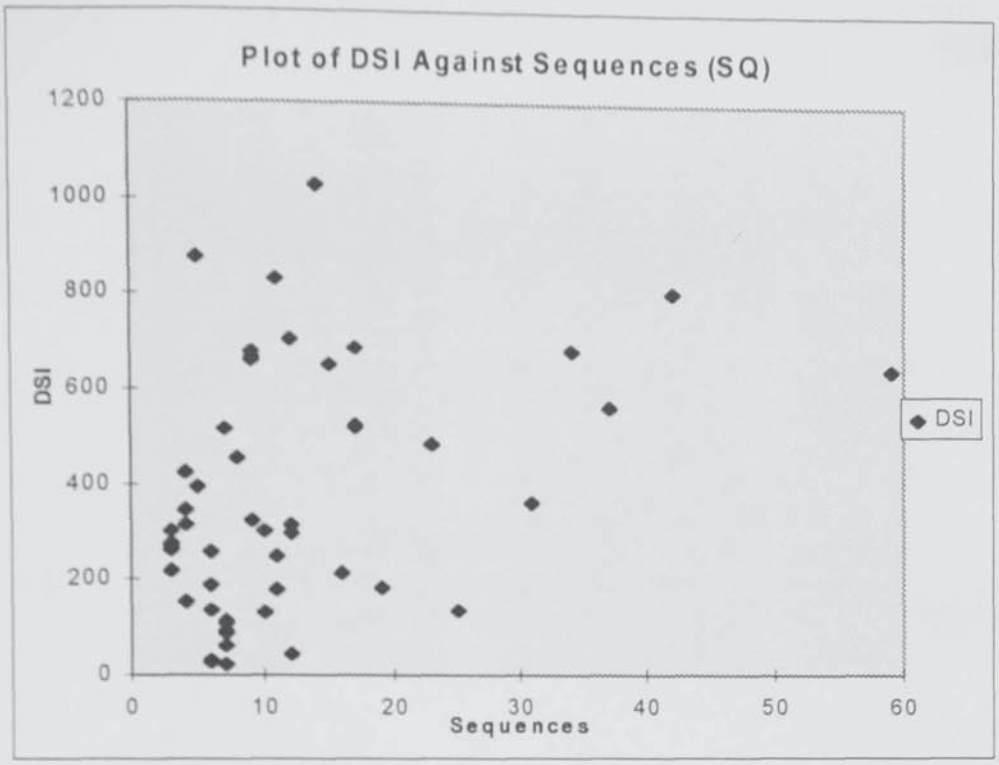
	Rating	AR	NS	PD	CO	IN
RELY	v low	0.80	0.80	0.80	0.80	0.60
	low	0.90	0.90	0.90	0.90	0.80
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.10	1.10	1.10	1.10	1.30
	v high	1.30	1.30	1.30	1.30	1.70
DATA	low	0.95	0.95	0.95	0.95	0.95
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.10	1.05	1.05	1.05	1.15
	v high	1.10	1.10	1.20	1.10	1.10
TIME	nominal	1.00 1.10	1.00	1.00	1.00	1.00
	high	1.30	1.10	1.10	1.10	1.15
	v high	1.65	1.25	1.25	1.25	1.40
	extra high		1.55	1.55	1.55	1.95
STOR	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.00	1.00	1.05	1.05	1.10
	v high	1.00	1.00	1.15	1.15	1.35
	extra high	1.00	1.00	0.98	0.92	0.92
VIRT	low	1.00	1.00	0.90	0.85	0.80
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.00	1.00	1.15	1.15	1.20
	v high	1.00	1.00	1.25	1.30	1.40
TURN	low	0.98	0.95	0.95	0.95	0.9
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.00	1.00	1.00	1.10	1.15
	v high	1.02	1.05	1.05	1.20	1.30
ACAP	v low	1.80	1.80	1.35	1.35	1.50
	low	1.35	1.20	1.15	1.15	1.20
	nominal	1.00	1.00	1.00	1.00	1.00
	high	0.75	0.75	0.95	0.90	0.92
	v high	0.55	0.55	0.75	0.75	0.85
AEXP	v low	1.40	1.40	1.30	1.25	1.25
	low	1.20	1.20	1.15	1.10	1.20
	nominal	1.00	1.00	1.00	1.00	1.00
	high	0.87	0.87	0.90	0.92	0.83
	v high	0.75	0.75	0.80	0.85	0.65
MODP	v low	1.05	1.05	1.10	1.25	1.50
	low	1.00	1.00	1.05	1.10	1.20
	nominal	1.00	1.00	1.00	100	1.00
	high	1.00	1.00	0.95	0.90	0.83
	v high	1.00	1.00	0.90	0.80	0.65

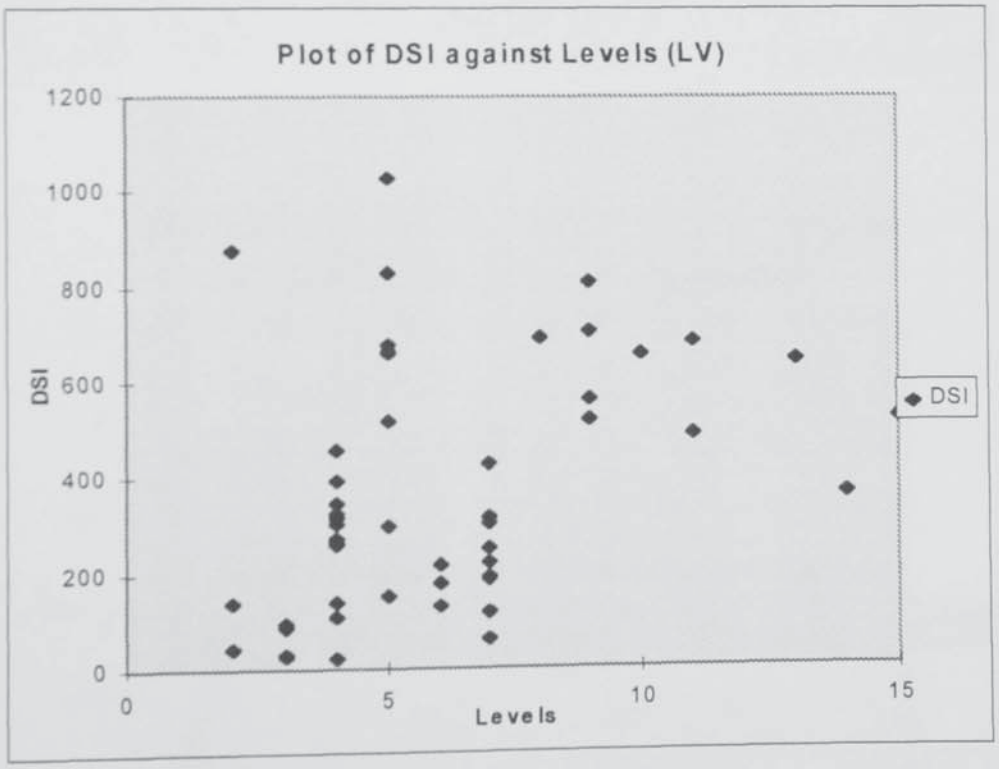
TOOL	v low	1.02	1.05	1.05	1.35	1.45
	low	1.00	1.02	1.02	1.15	1.20
	nominal	1.00	1.00	1.00	1.00	1.00
	high	0.98	0.95	0.95	0.90	0.85
	v high	0.95	0.90	0.80	0.80	0.70
SCED	v low	1.10	1.25	1.25	1.25	1.25
	low	1.00	1.15	1.15	1.15	1.10
	nominal	1.00	1.00	1.00	1.00	1.00
	high	1.10	1.10	1.10	1.00	1.00
	v high	1.15	1.15	1.15	1.05	1.05

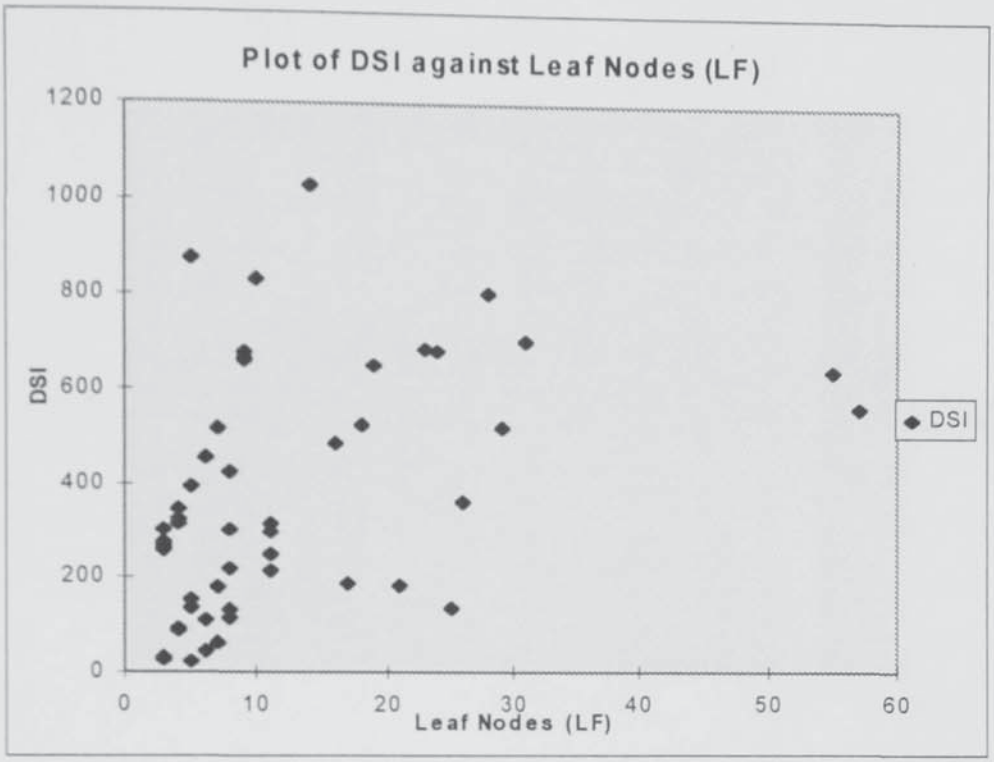
Scatter Plots of the Original Data

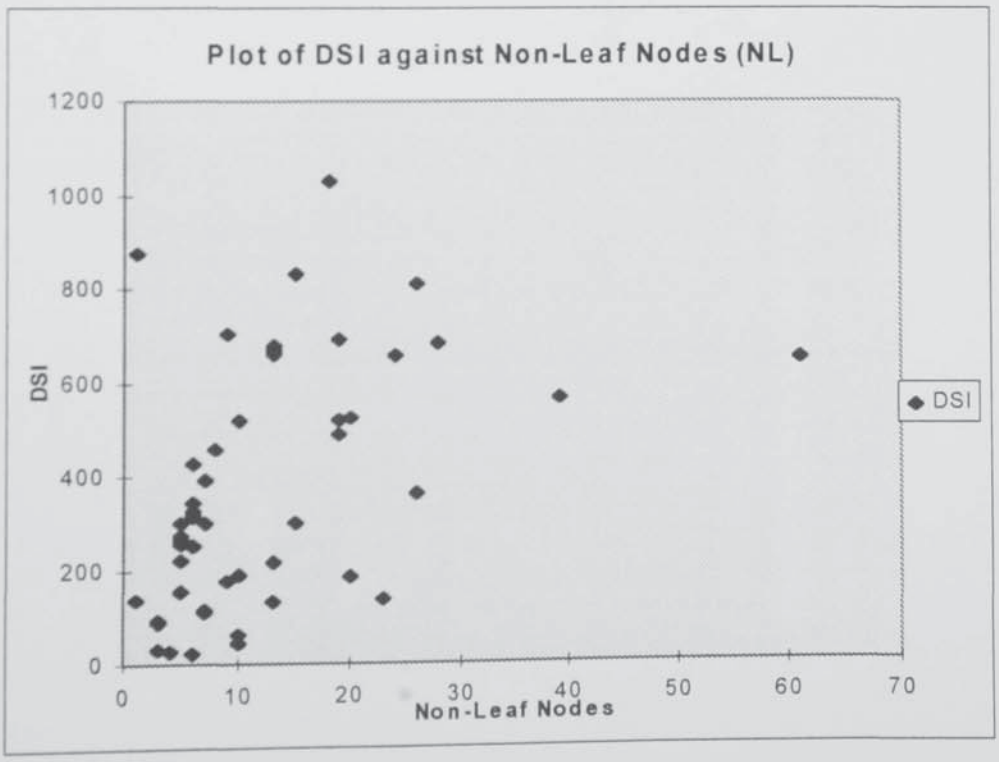


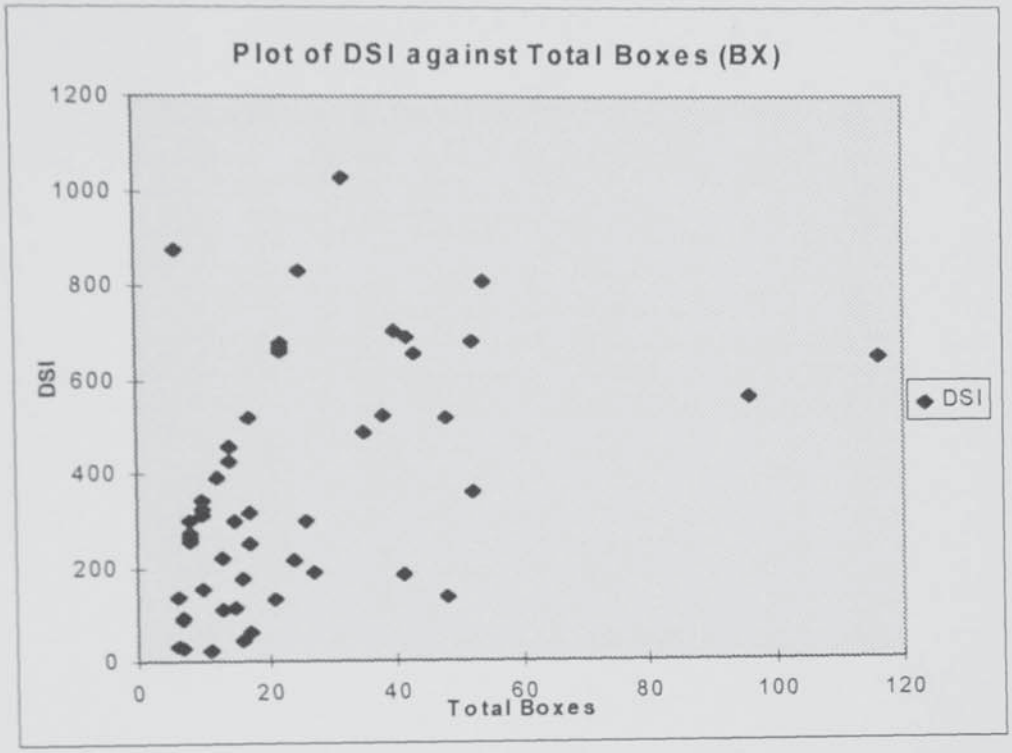



Plot of DSI Against Sequences (SQ)
1200
1000
800
600
400
200
0
DSI
0
10
20
30
40
50
60
Sequences
DSI


Plot of DSI against Levels (LV)
1200
1000
800
600
400
200
0
DSI
0
5
10
15
Levels
DSI


Plot of DSI against Leaf Nodes (LF)
DSI
Leaf Nodes (LF)
DSI
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60


Plot of DSI against Non-Leaf Nodes (NL)
DSI
Non-Leaf Nodes
DSI
0
200
400
600
800
1000
1200
0
10
20
30
40
50
60
70


Plot of DSI against Total Boxes (BX)
1200
1000
800
600
400
200
0
DSI
0
20
40
60
80
100
120
Total Boxes
DSI