A GENERIC MODEL FOR SOFTWARE SIZE ESTIMATION BASED ON COMPONENT PARTITIONING

A dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Software Engineering

June Marguerite Verner

1989

ABSTRACT

Software size estimation is a central but under-researched area of software engineering economics. Most current cost estimation models use an estimated end-product size, in lines of code, as one of their most important input parameters. Software size, in a different sense, is also important for comparative productivity studies, often using a derived size measure, such as function points. The research reported in this thesis is an investigation into software size estimation and the calibration of derived software size measures with each other and with product size measures.

A critical review of current software size metrics is presented together with a classification of these metrics into textual metrics, object counts, vector metrics and composite metrics.

Within a review of current approaches to software size estimation, that includes a detailed analysis of Function Point Analysis-like approaches, a new classification of software size estimation methods is presented which is based on the type of structural partitioning of a specification or design that must be completed before the method can be used. This classification clearly reveals a number of fundamental concepts inherent in current size estimation methods. Traditional classifications of size estimation approaches are also discussed in relation to the new classification.

A generic decomposition and summation model for software sizing is presented. Systems are classified into different categories and, within each category, into appropriate component type partitions. Each component type has a different size estimation algorithm based on size drivers appropriate to that particular type. Component size estimates are summed to produce partial or total system size estimates, as required. The model can be regarded as a generalization of a number of Function Point Analysis-like methods in current use. Provision is made for both comparative productivity studies using derived size measures, such as function points, and for end product size estimates using primitive size measures, such as lines of code. The nature and importance of calibration of derived measures for comparative studies is developed. System adjustment factors are also examined and a model for their analysis and application presented. The

model overcomes most of the recent criticisms that have been levelled at Function Point Analysis-like methods.

A model instance derived from the generic sizing model is applied to a major case study of a system of administrative applications in which a new Function Point Analysis-type metric suited to a particular software development technology is derived, calibrated and compared with Function Point Analysis. The comparison reveals much of the anatomy of Function Point Analysis and its many deficiencies when applied to this case study. The model instance is at least partially validated by application to a sample of components from later incremental developments within the same software development technology. The performance of the model instance for this technology is very good in its own right and also very much better than Function Point Analysis.

The model is also applied to three other business software development technologies using the IFIP[1] standard inventory control and purchasing reference system. The purpose of this study is to demonstrate the applicability of the generic model to several quite different software technologies. Again, the three derived model instances show an excellent fit to the available data.

This research shows that a software size estimation model which takes explicit advantage of the particular characteristics of the software technology used can give better size estimates than methods that do not take into account the component partitions that are characteristic of the software technology employed.

[1]International Federation for Information Processing

Acknowledgements

I would like to thank Professor Mark Apperley, my chief supervisor, for his encouragement and advice, and Mr. Ormond Tate and Mrs. Judith Holland of the New Zealand Correspondence School for their co-operation and help in the collection of data. Without this data the development of the ideas in this thesis would not have been possible. I would also like to thank Mr. Richard Hayward and Mr. Barry Jackson for their interest and encouragement as the work proceeded. Finally I would like to thank my mother who has always believed in educating women, and my husband Graham for his infinite patience.

TABLE OF CONTENTS

CHAPTER 3

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION TO SOFTWARE SIZE ESTIMATION

Software size estimation is an important but under-researched area of software engineering economics. The derivation of an appropriate size estimate is neither straightforward nor trivial, the process of sizing software is still subject to a wide margin of uncertainty and there has been comparatively little research on software sizing models to date [DACS87]. This is particularly true in the general area of business applications. The prediction of the size of a product as early and as accurately as possible is an elusive goal and currently "expert sizing depends on so many subjective factors that different 'experts' can arrive at radically different estimates. This underscores the need for more objectively based sizing techniques" [CONT86].

Software size estimation is important because size is a major cost driver in software development and hence is typically a critical component of early effort estimation [WANG85]. Estimated system size is used as input to cost and scheduling models so that development effort can be estimated and progress can be monitored throughout the development. For detailed scheduling estimates subsystem and component sizes are also necessary.

Escalating software development costs have led to the use of tools to increase developer productivity. There is a corresponding need to measure changes in productivity as the development environment changes (technology productivity). Productivity is usually defined in terms of output per effort unit with software size being the most commonly used measure of the output produced. Such a definition is adequate for developer productivity, but a different approach is needed for technology productivity. Software size in relation to software cost, schedule, and both developer and technology productivity is discussed further in 1.1 and 1.2.

There is a serious and continuing shortage of suitable development data for research into software size estimation models and productivity studies. A substantial and on-

going commitment from management is required for any kind of realistic data collection. Small programs written by students are not adequate for this type of research. To obtain enough relevant data long periods of time are normally involved (several man-years). The collection of project data can be quite time consuming and difficult without instrumented tools. The author was fortunate in having access to a large amount of high quality data from the New Zealand Correspondence School development.

With the emergence of an ever-increasing variety of new methods and tools for software development, together with evolutionary changes in existing methods, there is a need for more flexible and adaptable methods of software size estimation, that can change as the software development environments they are applied to change. Generic software size estimation models which can be tailored to particular environments are required to meet changing needs of this complexity.

1.1 IMPORTANCE OF SOFTWARE SIZE ESTIMATION AS INPUT TO COSTING AND SCHEDULING MODELS

With the continuing escalation of software costs and the cost overruns, schedule delays, poor reliability, or even project failure, that are typical of many software projects [DEMA82], the importance of software project control is obvious. Project planning has been identified as one of the critical problems faced by software managers [THAY81]. However, successful project planning relies on a reasonably accurate assessment of the effort required for completing the project [WANG84] together with the schedule options that may be available [JEFF87]. Software costing and scheduling models are being used increasingly to assist with the planning of software development. Costing models are used not only to predict the cost of developing the software but also to estimate the number of personnel that will be required for that development, the elapsed time to complete the project, and the time that will be needed to complete each phase of the project. Over the past several years a number of effort/costing models have been developed [DOTY77, WALS77, PUTN78, FREI79, BOEH81, RUBI85, JONE86, CONT86, JEFF87]. Many of these models are also discussed in [BOEH81, MOHA81, BOEH84, KITC85, CONT86, JONE86, KEME87]. These models, which all use software size as a major cost driver, are finding increasing commercial acceptance, especially in the

United States. One of the models, the Intermediate COCOMO model [BOEH81], when applied to a sample of 63 projects, gave results which were within 20% of actual cost 68% of the time. Another model, SPQR/20 [JONE86], is claimed to have been calibrated against historical data from a broad spectrum of programs and systems and to have generally come within ±15% of the observed results, except at the extreme ends of the model's operational range. Although there is a need for software sizing models with a reliability at least as good as that observed for these costing models, there has been comparatively little research to date in this area and variations of over 300% in the size estimation of a project have occurred during a project's development [SINC87]. It is important to remember that a cost estimation model cannot give a good cost prediction unless the estimated size which is used as input is close to the actual size. Many of the accuracy claims for cost prediction models are based on data sets which include size data from already completed software and all of the models have been developed using data obtained from completed software projects. In fact the COPMO model [CONT86] was developed using the same data that was used to develop COCOMO [BOEH81]. As a result, the accurate estimation of the size of proposed software was not a problem in these cases since the size was already available. However, after-the-event analysis of the relationships between cost drivers and effort, using a known size, though providing useful modelling data, does not address the size estimation question.

Software size estimation is the weakest link in the software cost estimation chain. It was included in Boehm's list of seven outstanding research issues in software engineering economics. He states that "The biggest difficulty in using today's algorithmic software cost models is the problem of providing sound sizing estimates". This difficulty is underscored by an experiment comparing six software size estimation models [DACS87] where the results of the experiment showed a range of estimates from 6,622 lines of code to 36,700 lines of code where the actual size was 9,177. Reifer [REIF87a p.285] states in his cost-estimating wish list that "better and more accurate ways of developing sizing estimates will be made available as research into function point theory begins to realize its potential". Itakura and Takayanagi [ITAK82] suggest that size, together with time required for software development, "is probably the most difficult aspect of any project". The accuracy of any size estimation model will depend on "where in the life-cycle the model is applied, the level and depth of the information available for the software system, the understanding of the model" being used "by the developing organization" as well as

"familiarity with the specific software application area. Only a certain level of accuracy and precision is possible at the early phases of the development where the level of knowledge about the software system is at a minimum. Any software sizing technique cannot be expected to compensate for an inaccurate understanding of what the software is to do" [DACS87]. Chapter 3 discusses current approaches to software sizing and recent research in some detail.

1.2 IMPORTANCE OF SOFTWARE SIZING IN PRODUCTIVITY STUDIES

The escalating cost of software, the accelerating demand for computer applications in recent years [WANG84] and the development of products that claim significant productivity benefits, have made productivity measurement in software development increasingly important. A survey of the productivity literature to 1982 was published in [PARI82] and Jones [JONE78, JONE81, JONE86] has also made an extensive study of programming productivity. He suggests that the term software productivity is generally used to imply either reduced calendar time for product development or reduced cost of that development. In order to calculate productivity it is necessary to have some unit with which to measure the quantity of product that has been developed. Jones states that "historically there has not been a workable definition for software yield. Programmer productivity has become an international concern and its three parameters are time, cost and yield". Yield is sometimes referred to by the terms result or output [KWON87]. Productivity has traditionally been measured in lines of code per programmer day (month or year) and a number of papers have been published giving productivity figures for specific languages [ALBR79, JEFF83, RUDO83, TATE87, VERN88], though increasingly dissatisfaction is being expressed with lines of code as a measure of yield, especially for fourth generation languages [RUDO83, JONE86, CANN86] and for comparisons of productivity between projects developed in different languages [ALBR79, BEHR83, RUD83, JONE86]. As a result there have been attempts [HALS77, ALBR79, DEMA82] to use units other than lines of code to measure the size of software. Some replacement metrics for measuring the output of the software development process that have been suggested are function points [ALBR79], BANG [DEMA82], and data base/diagram component counts [KWON87]. Chapter 2 classifies software size metrics and presents a critical evaluation of the more commonly used metrics.

1.2.1 Job Sizing for Productivity Studies

Software productivity studies are most commonly done for one of two reasons:

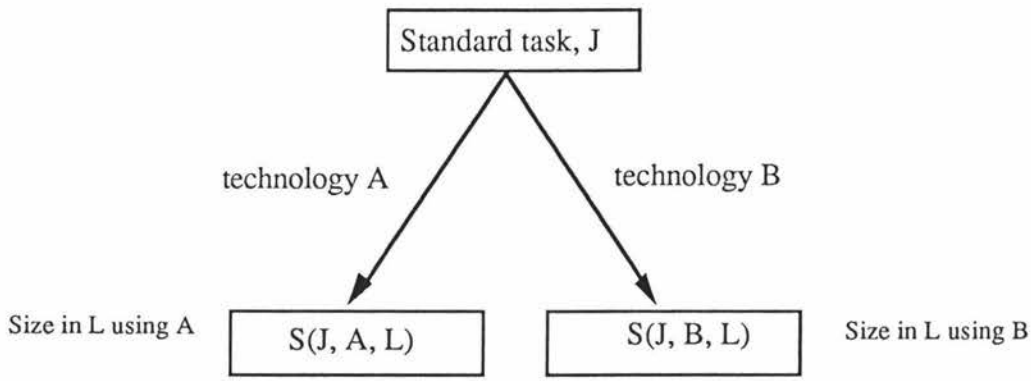
(1) to measure developer and/or managerial effectiveness within a defined software development environment (developer productivity)

(2) to measure the effect of different development methodologies, tools or environments on software production (technology productivity).

Software size measures are needed in both cases. However the sizes that need to be measured are different. In the first case the size of the product is of concern and the productivity unit commonly employed is lines of code per person-day. In the second case, however, there is as much interest in the size of the job to be done as there is in the size of the software produced using a particular development environment. The size of the job to be done should not be measured in technology-dependent terms (for example, lines of code) since this would defeat the purpose of measuring the technology-dependent effects. A measure which is as technology-independent as possible is needed for the second type of productivity study. Function Point Analysis (FPA) [ALBR79, ALBR83, ALBR84] is the most commonly used method that attempts to supply such a measure. See 2.5 for a critical evaluation of FPA.

It is considered appropriate here to examine more closely the role of software size measures in productivity studies which compare different software development technologies.

There are, in general, two approaches to this problem, the standard task approach and the standard measure approach. These are illustrated in Figures 1.1 and 1.2.



A and B can be compared directly using the ratio
S(J, A, L) : S(J, B, L)

Figure 1.1 The Standard Task Approach to Software Productivity Measurement

In the standard task approach a standard system is defined and then implemented in two different technologies. The sizes of the resulting systems can be directly compared and the comparative effects of the technologies on size will be obvious.

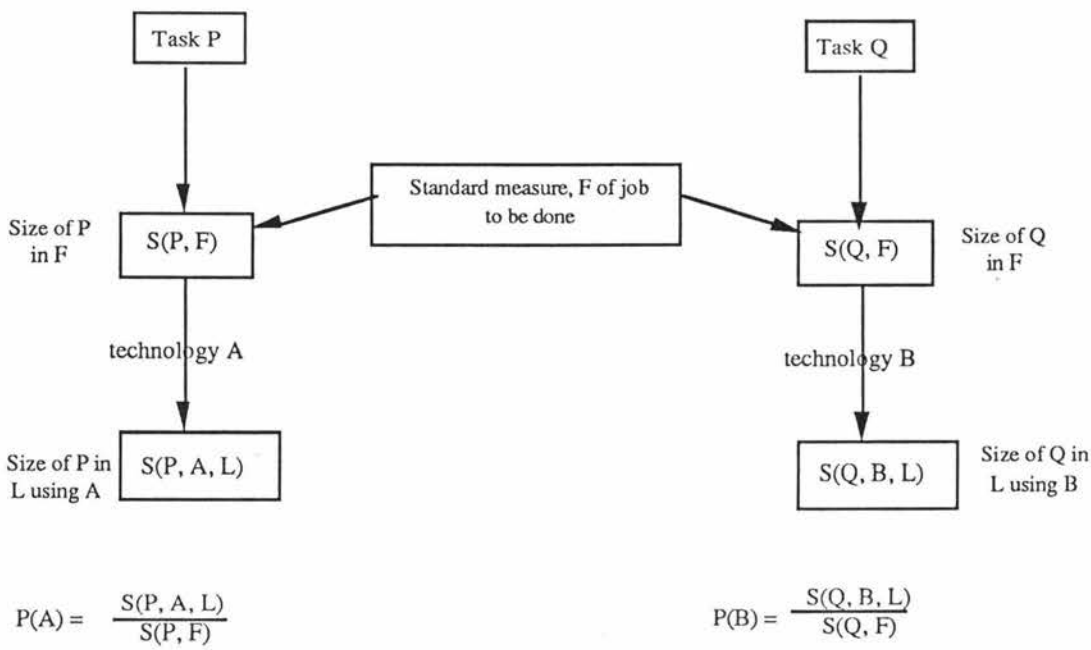


Figure 1.2 The Standard Measure Approach to Software Productivity Measurement

In the standard measure approach the sizes of the jobs to be done are estimated in a 'technology-independent' metric. The sizes of the completed jobs are measured in L (probably lines of code). P(A) and P(B) are then calculated by dividing the sizes of the jobs in L units by the estimated sizes in F units. Productivity comparisons can now be made because the dimensions of each are L units/F units.

The standard measure approach is generally used for several reasons:

(1) it is not usual for two different implementations of an identical system to be developed

(2) applications chosen to fill the standard task role are usually small, and therefore are likely to be unrepresentative

(3) the standard measure approach is usually very much cheaper than the standard task approach

(4) it is difficult in practice to keep all other factors constant when using the standard task approach.

A major problem, however, for software productivity measurement is that there is no generally accepted or completely satisfactory measure of the size of the job to be done. Function points are used as a *de facto* common measure particularly for business applications, but their use in this role has recently been criticized [SYMO88, VERN89]. The role of calibration for productivity comparisons is described in Chapter 4 and some examples of calibration for this purpose are included in Chapters 5 and 6.

The treatment of technology productivity above has been limited to sizing issues. In addition to the sizes of the products of different technologies, it is, of course, usual to compare cost, effort and duration as well. These matters are, however, outside the scope of this thesis.

1.3 OBJECTIVES AND STRUCTURE OF THESIS

The main objectives of this research are:

(i) by means of a critical survey of recent research in software size estimation to gain a greater understanding of both software size estimation methods and metrics

(ii) based on this understanding, to construct a generic software size estimation model that meets most sizing purposes and overcomes most size estimation problems

(iii) to test the model in a realistic environment .

The structure of this thesis follows these general goals and is divided into seven chapters as follows:

1. The importance of software size estimation and measurement for input into costing and scheduling models and also for job sizing in productivity studies. This topic has been examined earlier in this chapter.

2. A critical examination of software size metrics and their suitability for different sizing purposes and in different software development situations.

3. A critical study of software size estimation methods together with an attempt to classify them in a more systematic manner than has been done previously in order to highlight their relationships and reveal more of their essential structure. The identification of those methods whose development and/or generalization is most likely to lead to new and substantially improved sizing models.

4. The development of a generic software size estimation model which

   (1) accommodates a wide range of different sizing purposes and sizing metrics

   (2) overcomes many of the criticisms of existing models

(3) spans much of the software life cycle, but concentrates on the specification and early design phases where early size estimates are most in demand.

(4) can be based on more objective aspects of software representations from specification through to code, but does not necessarily exclude subjective aspects

(5) distinguishes metrics which are highly technology-dependent from those that are less so, relates them through technology-dependent factors, and provides a calibration mechanism for metrics with low technology-dependence (job-size metrics)

(6) builds upon a historical data base of technology-dependent data and includes an adaptive mechanism to cope with a shift or change in software technology, including recalibration of job-size metrics where necessary

(7) can be tailored to specific development environments or technologies for greater accuracy, or can be kept more general in applicability, possibly at some cost in accuracy

(8) allows for partial sizing to meet a variety of sizing purposes

(9) includes an adjustment factors model which characterizes and classifies adjustment factors effectively and provides for flexibility in their use to meet different sizing purposes.

5. The development and testing of an instance of the generic software sizing model for a large system of related data-centred business applications for which a significant body of data is available.

6. A demonstration of the applicability of the generic model to several different software development technologies applied to a single standard business system of small but significant size.

7. A summary of conclusions from this research and an outline of some areas for further research.

Data used for development and application of the model in Chapter 5 is included in Appendices A and B, while data used for the development of the model instances in Chapter 6 is included in Appendix C.

The terminology employed in the software economics area presents many difficulties, owing mainly to the many different concepts used in many different situations and the dearth of suitable commonly used words to describe them. As a result many different writers use different terms, and it has been necessary to compile a glossary of terms.

CHAPTER 2

A CRITICAL REVIEW OF PREVIOUS WORK ON SOFTWARE SIZE METRICS

Software size metrics are fundamental to all software size estimation. Since they are so fundamental they are described in detail in this chapter while the next chapter reviews the closely related topic of current approaches to software size estimation. This chapter presents a critical review of previous work in the area of software size metrics which is examined in relation to the ideas for software size estimation developed in this thesis. Because it has not been possible to completely separate some of the metrics from the methods of obtaining them, some discussion of methods is also included where appropriate.

There are problems with the measurement of software. Although many researchers are dissatisfied with the most commonly used metric, the line of code (LOC)[1], as the basis for measurement [ALBR79, RUDO83, JONE86 p.5, LEVI86] there is no commonly agreed or generally satisfactory substitute for it. Some suggested replacements to date include function points [ALBR79], length and volume [HALS77], character counts [BLUM86], token counts [LEVI86], data base component counts and diagram counts [KWON87], BANG [DEMA82], and metric vectors [BASI88] (though the last is not a single metric). Different software size metrics tend to relate to different aspects or concepts of system size such as the size of the product as against the size of the job to be done.

An approach to measurement that will give a consistent (usable at all stages of the lifecycle) size metric or set of related metrics, early enough in software development to be useful, is critical for software engineering economics. DeMarco [DEMA82] defined the qualities of a good metric in the context of software size measurement and estimation, as being measurable, independent (objective), accountable, precise, consistent and available early enough to be useful. Levitin [LEVI86 p.314] discusses the requirements of a basic unit for the measurement of software and suggests that the usefulness of a metric may be relative, as the same metric may be quite appropriate in one situation and useless in another. He suggests that not only should a metric be

[1] LOC is used as an abbreviation for line or lines of code depending on the context.

meaningful in the situation in which it is used, but that it should also have the following four properties:

(a) a clear and unambiguous definition so that it can be calculated algorithmically (i.e. be automatable)
(b) a relationship with an intuitive idea of program size; it should always have a positive value and be additive
(c) application to a wide variety of languages (i.e. be universal)
(d) a correspondence to some traditional means of measuring the size of printed material in natural language.

2.1 CLASSIFICATION OF SOFTWARE SIZE METRICS

Software size metrics can be classified into four main groups:

a) textual measures which are analogous to commonly used measures of the size of natural language in printed form
b) object counts of specification objects, or of objects occurring in other descriptions of software, or within the software itself
c) vector metrics which count separately more than one kind of object within the software or within some description of it, such as requirements, specification, and design documents or machine readable records
d) composite metrics which are single values produced by the application of functions to more than one kind of object count; these can be regarded as functions of vector metrics.

The characteristics of these four classes are described in sections 2.2 - 2.5. The most discussed and/or most used software size metrics fall within three of these groups, textual metrics, object counts and composite metrics. Figure 2.1 illustrates this classification and shows software size metrics within this structure.

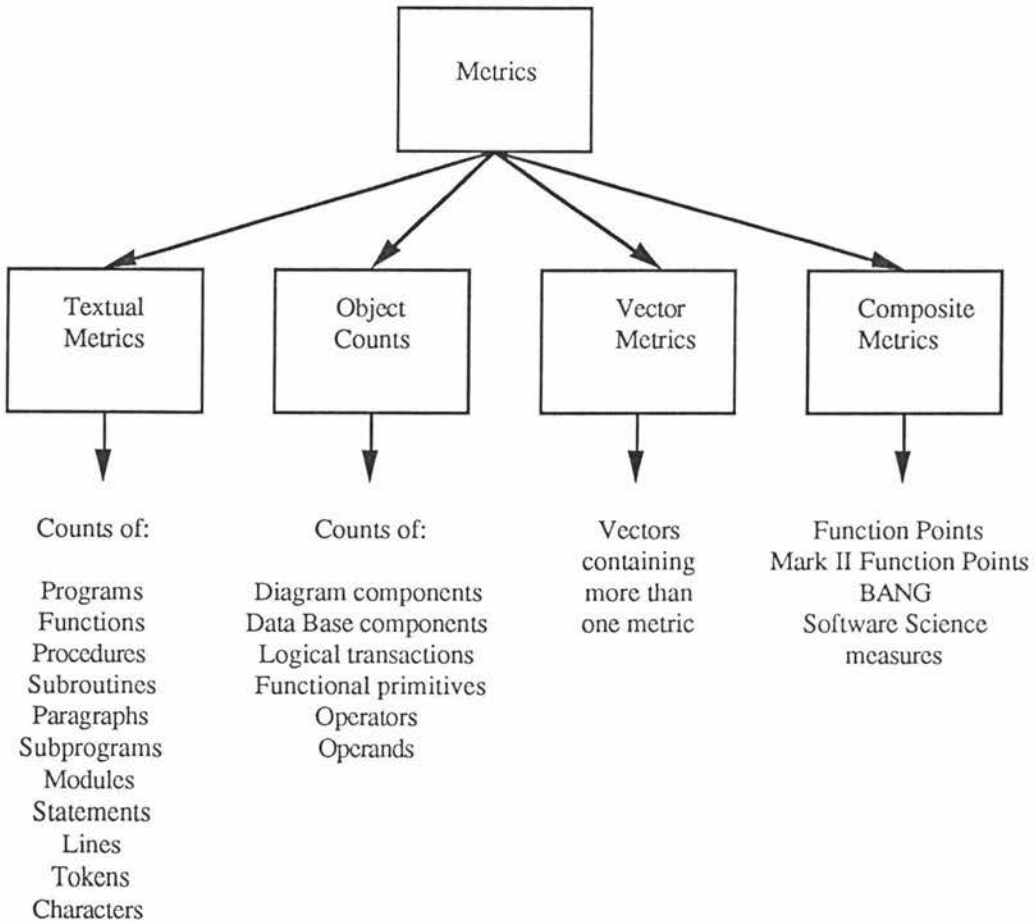


Figure 2.1 Structural Classification of Size Metrics

2.2 TEXTUAL METRICS

Textual metrics can be considered to be analogous to the traditional measures used with natural language, i.e. chapters, pages, paragraphs, sentences, lines, words and characters. LOC, statement counts, and token counts are textual metrics that have been used most commonly as software size metrics.

Chapters may be considered analogous to programs and at a crude level the number of programs making up a system may be of interest; however, this measure may not be the ultimate metric used but may be converted to a LOC measure by using the average LOC per program. If a particular environment has standards that result in the production of programs of a similar size the measure may be meaningful but if these standards do not exist, the metric may not be very useful.

Paragraphs have been considered analogous to modules, functions, procedures, subroutines, subprograms or paragraphs (in the COBOL sense). The modules used in experimental work done by Basili [BASI85 p.2] were defined as independently compilable units such as FORTRAN functions, subroutines and BLOCK DATA, or separately definable and retrievable units from an on-line library. Conte et al [CONT86] define modules as consisting of one or more functions and suggest that counting functions may be more useful than counting modules as the variation in LOC for a function may not be as great as that of a module [p.43]. They cite indirect support for this in [BASI79]. They also suggest [p.42] that, unless there are strict guide-lines on the way programs are divided up, this metric will give little information. It is unlikely that modules, procedures, subprograms, etc. will be of any real use unless they are associated with another metric such as LOC and Levitin [LEVI86 p.314] suggests that, although modules, functions, procedures etc. may have been used as size metrics, they are clearly too coarse for most applications.

2.2.1 Lines of Code

The LOC is a textual unit that can be considered analogous to a line of text in the traditional measures associated with natural languages. Dunsmore [DUNS84] believes that, if one uses a standard definition consistently, LOC is probably the most intuitive measure of program size. The LOC is the oldest and most commonly used metric for software size and is still extensively used today. This metric originated from the number of punched cards that it took to contain a program [CONT86 p.32], and appeared in the literature as early as 1969 [ARON69], although clearly it had already been in use for some time before then. Levitin [LEVI86 p.314] suggests that the dominance of this metric results from its simplicity and ease of application together with "its naturalness for such older computer languages as ASSEMBLY and FORTRAN and to some degree the inertia of tradition". For some languages, for example, BASIC and FORTRAN, there is usually a one-to-one correspondence between a LOC and a statement. However, for many modern languages this is no longer true [LEVI86 p.314]. Basili et al [BASI83 p.658] described LOC as "a more primitive volume metric".

Levitin [LEVI86 p.315] states that LOC is more a measure of the size of a program's representation rather than of the size of the program. Jones [JONE86 p.63] suggests that although the problems of using LOC as a measure are severe, the concept is so deeply embedded in the programming industry that it cannot be quickly replaced in

the short to medium term. With the introduction of Ada, a language that fits relatively easily with a LOC count, a LOC metric may be with us for many years to come. Using Levitin's [LEVI86] criteria LOC are automatable, additive, can have definition clarity, are nearly universal in their application to traditional programming languages but may not be quite so widely applicable in future, for example with some Computer Aided System Engineering (CASE) tools.

A brief discussion of some of the problems that can arise with the LOC as a metric is presented in the next section. For a fuller discussion of these problems the reader is referred to [JONE86].

(i) Problems with the Lines of Code Metric

Jones [JONE86 p.15] states that, although the phrase LOC is used almost daily, there is no universally agreed definition of precisely what a LOC is. He suggests that there are eleven major variations in software line-counting and that these variations fall into two different groups, both of which must be considered. The first group concerns line-counting variations at the *program* level while the second concerns line-counting variations at the *project* level. Differences in counting techniques can result (in extreme cases) in differences of more than five to one at the program level [JONE78].

LOC was initially used for counting lines of machine code or assembler code at a time when a LOC was more easily identified because of its fixed format. Once free-form assembly languages were introduced, more than one statement could be written on a line. This signalled the beginning of problems for the LOC as a software size metric [JONE86 p.47]. More difficulties appeared with the introduction of third generation languages and still others have emerged with the development of fourth generation languages (4GLs) and CASE (Computer Aided System Engineering) tools. The difficulties with LOC as a measure are not restricted to a concern with software sizing issues *per se*, but also extend to significant problems with productivity measures. These latter difficulties will not be discussed in any detail here. The reader is again referred to [JONE86] for a full discussion.

(a) *Source or Object Code*

Insofar as LOC are used as a size measure for the actual product developed or delivered to a customer, the problems that arise are caused by the lack of, or even impossibility of, a definition for a standard counting method that can be used across

all languages. Counting may be based on lines of source code or alternatively on lines of object code although most, but not all, published work deals with source code. If, instead of source code, object code is used other considerations arise. What effect will optimizing compilers have on the number of lines of object code? Arthur [ARTH85] suggests that an optimizing compiler produces 25% fewer statements than a normal one. It may be simple to count object lines in a *completed* program but it is not easy to estimate the final compiled size of a program before it is written in a high level language. Ratios of object code per source instruction are reported to vary from 1.6 : 1 to 6 : 1 [JONE78 p.11] for the same language. In [JONE86 p.49] a table of ratios of source statements to executable statements for a number of languages shows variations from 1:1 for basic assembler to 1:50 for spreadsheet languages.

(b) ***Statement Counting***

The coding standards within an organization, and between organizations, may vary significantly, with some organizations or programmers using several statements per line, or spreading one statement over several lines, while other organizations may enforce a standard one-statement-per-line regime. A line may contain more than one statement but Boehm [BOEH81 p.59] and Conte, Dunsmore and Shen [CONT86 p.35] consider a line is a line, no matter how many statements it contains. Boehm [BOEH81 p.59] however uses the term delivered source instructions (DSI) which he defines as LOC or card images, as an alternative to LOC.

(c) ***Data Definitions and Non-executable Code***

One variation is to count only executable lines. Systems written in some earlier languages, for example FORTRAN, did not use the large file and data definitions that other languages may require. Programs written in COBOL, however, may have many pages of non-executable code. There have been discussions on whether all or any of the COBOL DATA DIVISION should be included in the LOC count. Jones [JONE86 p.20] considers that data definitions are elements of LOC and Duncan [DUNC88] also includes data declarations in a LOC count. Dunsmore [DUNS84] agrees and states that because effort has to be expended on all source statements (including declarations), and not just those that will be executed, they should all be included in the LOC count. Boehm [BOEH81 p.479] stated that he included only one third of the data division from the COBOL systems that were part of the data base that he used as the basis for the building of his COCOMO software costing model. The other programs in his data base were mainly written in FORTRAN and Assembler.

(d) *Reused Code*

It is also possible that some or all of the data division in a program may be made up of code copied from a library, another program or even from another system. Similar difficulties occur with included or reused code of all kinds. Should all of it, or only some of it, be included in the final LOC count? [JONE86 p.20] takes the view that the code delivered to the user should be the basis of the count and included code is counted every time it is included. This view is discussed more fully in section 2.2.1 (ii). Other writers [JEFF79] do not take this stance and count included code only once together with the relevant copy or include statements.

(e) *Comments, Blank Lines, JCL and Scaffolding*

Some writers [WALS77, BAIL81, MAST85, DUNC88] have included comments in their count of LOC and others have possibly included blank lines by counting pages and multiplying them by the number of lines on a page. However, most exclude both comments and blank lines [BOEH81 p.59, BOYD84, CONT86 p.35, JONE86 p.20]. Some writers [BOEH81 p.59, WALS77] include Job Control Language (JCL), while others [JONE86 p.20] exclude it. Some [JONE86 p.20], exclude non-delivered software, such as scaffolding, while others [BOEH81 p.58], suggest excluding it unless that software has been developed with as much care as the delivered software. Many writers have limited their discussion to programs rather than systems, and therefore do not mention their attitude to inclusion or exclusion of lines of JCL and scaffolding.

(f) *Development Language*

With the many different languages, and dialects of languages, that are used for software development, and the different counting methods used across installations, it is often impossible to compare program size and productivity figures between installations. Some of the published data omits the precise basis of the LOC measurement for a particular system and, without this, it is impossible to compare the sizes of different systems even if they are written in the same language. It may require several lines of assembler language to provide the functionality of one line of COBOL, and many lines of COBOL to provide similar functionality to one 'line' of a 4GL, for example ALL, PowerHouse, LINC, Informix. The introduction of 4GLs has meant that the existing problems with the LOC as a software size metric are exacerbated. LOC, as they are generally known, may not exist in some programs written in these languages, for example where techniques like form-filling are used for the non-procedural specification of parts of a system. As a result many 4GLs

must have their own line counting conventions defined. The conventions used to count non-procedural code may be quite different from those required to count procedural code. The procedural code will probably be similar to conventional third generation languages and hence rather easier to count. A counting convention for one 4GL [VERN88] is discussed in section 5.1.

(ii) Approaches to Solving the Problems of Lines of Code

Approaches that can be used to resolve the major problems discussed above depend on the measurer's view of the software and the purpose for which it is being measured. If a size estimate is being made in LOC as input for costing, then a standard approach, of which the following is an example, would appear to be necessary to achieve consistency:

a) Using only a particular version of a standard language and system, though this may not always be possible.
b) Enforcing standard formats for the language and using a "pretty printer" to guarantee a standard layout and, preferably, to also count the LOC as they are developed.
c) Treating included code in the same way as subroutines or macros and counting them only once, together with their 'calls'.
d) Adopting a similar principle to (c) in regard to reuse.
e) Having a standard approach to the inclusion or exclusion of scaffolding, JCL, comments and blank lines.
f) In accordance with more modern approaches to programming languages, counting all declarative code.

Such an approach would, in some instances, reduce many of the current problems of measuring software size. It would still be difficult to compare the sizes of programs written in different languages, and systems written in some 4GLs would also be difficult to deal with. Jones [JONE86 p.59] has suggested that the language problems may be overcome by normalizing the different languages into basic assembler language. This involves converting the actual program size into equivalent assembler lines by multiplying the number of source statements by the nominal level of the language. He states that although the method is statistically unsound, and obviously somewhat subjective and unreliable, it is at least a starting point. He suggests language levels for a number of the currently used languages and has

provided a form of language size conversion in his products, SPQR/20 and SIZER/FP (although this is in a slightly different context and is discussed in more detail in section 3.6). Language expansion factors are used in a similar fashion, in some other size estimation products, for example, ASSET-R [REIF87], QSM FP [QSM87] and Before You Leap [GORD87].

However, if the area of interest is the amount of code that is actually delivered to a customer, then a different approach to resolving some of these problems, possibly one similar to that used by Capers Jones [JONE86 p.20], may be appropriate. This approach applies the following conventions:

a) Lines are terminated by delimiters.
b) Verbs or operational statements are included.
c) Data definitions are included.
d) For Assembler language, macro expansions are included.
e) For COBOL, included code is counted every time it is included.
f) The code delivered to the user is the basis for the count.
g) Comments are excluded.
h) Job control language is excluded.
i) Temporary code developed to aid testing is excluded.

The emphasis is on programs that are actually delivered to the end users, rather than on the development of that code, and the measures are aimed at what users receive, not what programmers do. This is a further illustration of a need to adapt the measure to fit the purpose of the measurement.

2.2.2 Statement Counting

The statement is a textual unit that can be considered to be analogous to a sentence in the traditional measures associated with natural languages, though in languages allowing nested statements it is perhaps analogous to a clause. Most programming languages have statements as their most prominent syntactic component. Levitin [LEVI86 p.315] suggests that statement counting, because it is a finer measure than LOC, overcomes the problems of counting LOC in free format languages. In [FEUE79], where PL/I programs were investigated, the number of statement clauses, which "roughly corresponds to the number of semi-colons" was used; Vosburgh et al [VOSB84] and Jones [JONE86 p.16] count statements rather than LOC. Basili et al

[BASI83] count both statements and LOC. Statements, however, like sentences, may be of dramatically different lengths [LEVI86 p.316]. Using Levitin's criteria statements have definition clarity, their counting is automatable, they are additive, but may not be universal across all languages.

There have, however, been a number of problems identified with statement counting. Levitin [LEVI86 p.315] noted the following difficulties with the use of the statement as the basic metric:

(a) It can be difficult to get a standard definition of a statement applicable to a wide variety of languages. Although most known programming languages are imperative or statement-oriented there are important languages such as APL, LISP and PROLOG which are not organized around the notion of a statement. APL and LISP are organized around expressions while PROLOG is organized around rules.

(b) Another problem results from the possibility of using structured or nested statements in languages like Pascal. The count of nested statements is a non-trivial task. One cannot overcome these problems by only counting statement delimiters, for example, the full stop in COBOL or the semi-colon in Algol-like languages, because these constructs may not actually coincide with the statements as defined by the language. Levitin gives an example of a single line of Pascal which has three statements (one IF, and two assignments), but only one semi-colon. Similarly a COBOL sentence may contain many verbs or statements. Indeed, sentences of a whole page occur in some COBOL programs.

(c) Levitin [LEVT86 p.315] also suggests that inconsistencies with the use of semi-colons in Algol-like languages (where they can be used to separate statements and for increments of an index variable in a FOR statement) are typical of the inconsistencies across or between programming languages.

2.2.3 Tokens

The token is a textual unit that can be considered as analogous to a word in the traditional measures associated with natural languages. In an effort to overcome some of the difficulties of using LOC or statements to measure program or system size Levitin [LEVI86] has suggested using tokens counts. He asserts that they are a superior metric because they are a finer unit of measurement and that, just as the word

is the fundamental unit for the English language, the token is the fundamental unit of programming languages. He also suggests that they have (almost) definition clarity, are countable automatically, are additive, are universally used across computer languages and correspond to natural language units. In view of this he is surprised that they have not been used in software engineering research and for practical applications.

Levitin [LEVI86 p.316] defines tokens as:

> "the basic syntactic units from which a program can be constructed. Each token represents a sequence of characters that can be treated as a single logical entity. At the same time, these entities are atomic, i.e. they have no further possible subdivisions."

Identifiers, numbers, strings and punctuation symbols are all typical tokens of programming languages. Levitin suggests that tokens have all the desirable properties of a size metric. He notes that some languages recognize the fundamental importance of tokens, for example, ADA defines a program as a sequence of tokens or lexical units. During lexical analysis, the first stage of compilation, programs are split into tokens which can be counted automatically. Programs written in some 4GLs, or parts of programs that have been developed with the use of form-filling techniques, are likely to be made up of nothing that is readily recognizable as a LOC [VERN88]. Token counts may be a more appropriate measure of the sizes of systems written in these languages.

There may be some difficulties with the use of tokens as a general measure. Any software of realistic size will possess many thousands of tokens. However as we use KLOC (thousands of LOC) why should we not use KTOK [LEVI86 p.316] ? There are also some problems with the definition of a token, or token pairs, for example, opening and closing brackets, begin...end statements etc.; should they count as one token or two? Levitin suggests that each be counted separately but in software science token counting [HALS77, FITZ78] these matching pairs are normally counted as a single token.

2.2.4 Character Counts

Characters in the English language can be considered to be analogous to characters in program text although this is not a commonly used metric. One of the difficulties with character counting is its sensitivity to the changing lengths of variable names. Using Levitin's criteria [LEVI86] character counting is automatable, has definition clarity, is additive and is universally used across computer languages, however it may be at such a low level that it might not always be useful. A character count divided by 30 was used to define a LOC in MUMPS programs [BLUM86].

2.3 OBJECT COUNTS

Other measures of software size are based on counts of various objects occurring within software or its specification. Objects that have been counted include *procedural* LOC, JCL statements (or lines), data declarations [BOEH81 p.59], unique operators and operands, total operators and operands [HALS77] functional primitives [DEMA82] and logical transaction types [SYMO88]. Albrecht and Gaffney [ALBR83] defined a metric based on the sum of the unique input and output items in programs. System BANG, described by DeMarco in 1982, is based on counts of functional primitives that are adjusted by the numbers of input and output items at their boundary while Data BANG is based on data base component counts [DEMA82]. Because BANG is a composite metric based on two types of object counts it is discussed in section 2.5. Kwong [KWON87] makes two similar but rather general suggestions. He states that components making up data base definitions can be counted and that this metric is useful for data-centred designs with simple procedural requirements and also that the diagrams or charts used with some development methods (for example Structure Systems Analysis [DEMA79], HIPO, flowcharts) allow developers to generate countable elements. Although diagram counting has not been used to any great degree to date, with the increasing use of CASE tools with graphical interfaces, metrics of this type may become more important in the future. None of the measures classified in this group as object counts appear to have been used on their own other than experimentally.

2.4 VECTOR METRICS

Work of general relevance in the area of software metrics has been done by Basili and Rombach (BASI88) in the TAME project. Among the important concepts they discuss is that of a characteristic vector of metrics for a development environment. They state that "most aspects of software products are too complicated to be captured by a single metric" and that "for both definition and interpretation purposes a set of metrics (a metric vector), that frame the purpose of the measurement, needs to be defined". They also state that we cannot just use "metrics from other environments. The models and metrics must be tailored for the environment in which they will be applied". These general concepts also apply to the more restricted subject of software size metrics. Conte et al [CONT86 p.78] suggest that a vector metric may be a more useful measure of the properties of software than a single measure although their discussion was in the context of complexity measures.

Most of the approaches that use object counts are based on counts of more than one object type and hence can be considered to be metric vectors although in practice they are often combined later into a single unit (see section 2.5). Function Point Analysis (FPA) [ALBR79] uses files in combination with input and output data elements or record types as its basic units. Symons [SYMO88] in MarkII uses input data elements, output data elements and entities for each logical transaction type. From the vector metric point of view the component measures of FPA and MarkII can be regarded as entries in a vector of metrics although in practice they are combined into a single composite metric. If both System BANG and Data BANG [DEMA82] are used to measure the same software then they can also be considered to be entries in a metric vector.

2.5 COMPOSITE METRICS

The application of suitable functions to more than one kind of object count, i.e. to a vector of counts, can result in single-valued composite metrics. The software science metrics [HALS77], together with BANG [DEMA82], FPA [ALBR79] and MarkII [SYMO88], are examples of composite metrics that have been used for software size.

System BANG and Data BANG [DEMA82, DEMA84] are composite metrics based on functions of object counts, i.e. functional primitives and operands, data base objects and relationships. Software science measures [HALS77] are based on four

counts, unique operator and operand counts and total operator and operand counts. Several single software science measures, length, volume and vocabulary are functions of these basic counts.

Both FPA and MarkII attempt to combine several basic counts into a single measure which is related to 'function value delivered to the user'. To obtain the single (composite) function point metric from a vector involves the resolution of the question of function value provided to the user. This tricky question is resolved by giving different components different weights. These weights are meant to reflect the relative value of a component to the user; however not all users have agreed with Albrecht's weightings [SYMO88] (see section 2.5.1(i) which examines this topic in more detail).

Although there are some similarities between System BANG, FPA and MARKII, System BANG does not have a fixed number of component or object types (functional primitives), FPA has five component or object types (inputs, outputs, files, interfaces and inquiries) while MarkII has a single component or object type (the logical transaction). Both FPA and MarkII are based on the user's (external) view of the system (ie they are functions of input and output data, and files associated with transactions that are visible to the user). System BANG on the other hand is obtained from both externally visible data elements and internal functional primitives. BANG is obtained from functions of input and output items and their associated functional primitive. Some of these functional primitives would not be visible to the user (for example, device and storage management).

Even some of the work that uses LOC can be viewed as dealing with a type of composite metric, for example, where the procedural lines in COBOL programs have been treated differently from data declarations [BOEH81 p.479]. Only one third of the data declaration lines, in COBOL programs, were counted for the COBOL systems used in the development of the COCOMO model. Some of the other work using LOC has also dealt with JCL differently. In some cases the JCL was excluded [JONE86 p.20] while in others it was included [BOEH81 p.59, WALS77].

2.5.1 Function Point Analysis

Function Point Analysis (FPA) was introduced by Albrecht [ALBR79] in 1979 and has been developed further since then [ALBR83, ALBR84]. A modified version of FPA, MARKII, was described by Symons [SYMO88] in an attempt to overcome problems with FPA that he identified. Reifer [REIF87a p.285] states in his cost-estimating wish list that "better and more accurate ways of developing sizing estimates will be made available as research into function point theory begins to realize its potential".

(i) Aims of Function Point Analysis

A function point [ALBR79], is a composite metric which aims to measure the function value that a system provides to the user. It is claimed to be independent of the language in which the software is written. This claim to language independence is discussed in greater detail in 2.5.1(iv) below. Albrecht [ALBR79, ALBR83, ALBR84] believed that the intrinsic size of a system was the product of a measure of the information that the system processed, and a technical complexity factor. The latter took into account the various technical and other factors involved in developing and implementing the information processing requirements (for example performance, ease of use, etc.) which influence the final size of the software. He introduced FPA to measure the size of a system based on these two factors. The unadjusted function points measure the information processed and the adjustment factor the technical difficulties of implementation.

The function point measure is a dimensionless number that is based on the end-user's view of the system [ALBR79]. Albrecht refers to this as "function value delivered to the user". There is great difficulty with this concept. In the calibration of MarkII, Symons [SYMO88] effectively equates function value with production effort or cost. However, other possible interpretations include:

(i) some kind of economic measure of information utility, though this would seem to apply more readily to outputs than to inputs
(ii) some measure of specification size - based on the idea that transactions that have larger and more complex specifications provide more function to the user
(iii) some product measure, again presumably related to value (or more likely cost) in most cases.

The function point measure was proposed with the aims of being independent of LOC and implementation language, being calculable early in the development cycle [ALBR83], and being able to isolate the intrinsic size of the system from the environmental factors [ALBR84]. Albrecht [ALBR84] also claims that nontechnical users can understand and evaluate the measure. An implied aim for the introduction of the measure is that it has an acceptably low measurement overhead [SYMO88].

(ii) Function Point Computation

The measure is determined from the number and complexity of five **component types**:

external input types
external output types
logical internal file types
external interface file types
external inquiry types.

This initial count is summed (unadjusted function points) and later modified using fourteen system adjustments to get a final function point count for the system (adjusted function points) [ALBR83, ALBR84, ZWAN84].

The Albrecht FPA model is thus:

$$F = T \sum_k F_k(C_k)$$

where

$$T = .01 \sum D_j + .65 \quad 0 <= D_j <= 5$$

for 14 general information processing adjustments D_j, and $F_k(C_k)$ is the raw function point measure for the kth component, calculated according to the following tables (Table 2.1 and 2.2), which show the component types, the items whose counts determine the raw function points of components, the class intervals for those item counts, and the weights for low(L), average(A) and high(H) function point levels for each component type. An example using these tables is given below.

Component type	Component FP determinants & class intervals											
	files			record types			data elements			level weights		
class ->	1	2	3	1	2	3	1	2	3	L	A	H
file				1	2 - 5	>5	<20	20 - 50	>50	7	10	15
interface				1	2 - 5	>5	<20	20 - 50	>50	5	7	10
input	<2	2	>2				<5	5 - 15	>15	3	4	6
output	<2	2 - 3	>3				<6	6 - 19	>19	4	5	7
inquiry	Uses files & data elements and takes greater of input or output level									3	4	6

Table 2.1 FPA Component Types, Class Intervals and Weights

The level weights are derived from the class intervals of the two relevant component FP determinants as follows:

		Component two class		
		1	2	3
Component one class	1	L	L	A
	2	L	A	H
	3	A	H	H

Table 2.2 Function Point Level Table

with a possible subjectively applied variation of one level up or down depending on an assessment of component complexity.

An example may help to clarify the use of the above tables. A file with 1 record type and 12 data elements is low (L) and thus rates 7 raw function points, whereas an input involving references to 4 files and 8 data elements has a high (H) level giving 6 raw function points.

The current FPA method, described above, is a little different from that which was introduced by Albrecht in 1979 [ALBR79]. The original work did not include an external interface type, all the component types had the same complexity and the adjustment factor for system complexity was ± 25%, not the later ±35%. The measure has been refined from time to time, as for example in the GUIDE Current Practices document [ZWAN84].

(iii) Use of Function Point Analysis

Since many of the system adjustment factors could be classed as cost drivers rather than size drivers, the final function point count can be considered to be more a measure of total effort than of size. Symons [SYMO88] refers to the metric as a size measure, but his usage of the term size is more in the sense of total effort for the job to be done, and does not adequately distinguish between size and effort.

Function point data is used within IBM to measure efficiency (FP/work month), quality (defects/FP), trends in productivity, and maintenance support (work hour/FP) [DACS87]. FPA was originally introduced by Albrecht to compare productivity between projects that were written in different languages and used different technology. He used the average number of LOC required to develop a function point to show the relative productivities of COBOL, PL/1 and DMS/VS [ALBR79]. Other writers [RUDO83b, REIF87, JONE86 p.77, GORD87, VERN88, VERN89, LIM89] have extended this work to give relative productivities for some other languages, including 4GLs, and the measure has been incorporated, sometimes with additional refinements, into several commercially available size and cost estimation tools [GORD87, JONE86, REIF87, QSM87]. These tools are discussed in more detail in section 3.2.2. Symons [SYMO88] suggests that the FPA method is likely to become a *de facto* industry standard in the business information processing industry but "before that happens it is necessary to examine the method for any weaknesses and to overcome them".

(iv) Criticisms of the Function Point Measure

Many of the problems with FPA as a measure of system size result from its history. FPA was originally introduced for productivity studies (including technology productivity) but is increasingly being used for size estimation. There is a conflict in the requirements of a metric with these two different uses. One requires technology independence, the other technology dependence. The main requirement for productivity studies is for a metric with a high degree of programming language independence, or more generally technology independence. By providing a measure of "function value delivered to the user" FPA has claimed this technology independence. On the other hand, to enable the product size in a target language to be estimated effectively a high degree of language or technology dependence is

necessary. This conflict has been resolved in the past by using technology-dependent multipliers, such as LOC per function point or effort in person-hours per function point, to convert from the (supposedly) technology-independent 'user function' measure (function points) to the technology-dependent size or effort measures of the target system.

The problems with the function point measure can be divided into three groups -- problems with the measurement of the unadjusted function point size, problems with the system adjustments and other general problems. Some of these difficulties appear to be the result of FPA having been developed in the late seventies with mainly batch systems and the use of some of the system adjustments to make up for the effects of more modern technologies.

(a) ***Unadjusted Function Point Measurement***

The problems that occur here include division into a technology-dependent set of component types (input, output, inquiries, files and interfaces), an oversimplified classification of component types, the concept of function value delivered to the user, an inadequately documented and apparently rather arbitrary choice of weights, an inability to deal effectively with internal processing complexity, some inherent subjectivity and summation inconsistencies.

Technology Dependence of Component Types

FPA is claimed to produce a technology independent, dimensionless value. The first question that must be asked is - are the user functions (component types) of input, output, inquiry, file and interface on which FPA bases its information processing size measure themselves technology-independent? Symons [SYMO88] maintains that they are not. He replaces files by entities, in an entity-relationship sense [CHEN77] in an attempt to update FPA in his MARKII version. In a relational context the files could be replaced by relations. An updating of FPA's division of functions into inputs, outputs and inquiries would also seem to be appropriate with today's technologies. Modern interactive systems tend to blur these divisions. For example many input transactions, particularly those involving update information, display the current state of relevant and related information both initially and during the transaction, thus combining inquiry with input.

If a new technology allows the user to do new things, or to do things differently, then it would appear that user function has changed with the technology. Its measurement must therefore change also. System components and measurement components

should match. Where technology changes, the metric vector of relevant object counts may need to change also. The original Albrecht component categories of file, input, output, inquiry and interface file do not always fit current technology well. Where this is the case they should be replaced by the natural components of the system model. Changes in system components are needed to reflect the characteristics of today's modern interactive systems which typically use batch methods almost solely for reports and have general purpose screen or window handling facilities for most other types of user-system interaction. The individual component metric vectors will then include elements of importance to that particular type of component. For example in the case discussed in [VERN89] these are: for menus, the number of choices; for screens, the number of data elements and files; for reports, the number of data elements, files and line types. For other technologies, the choice of components and component vector elements will be different again [VERN89]. For example, in an object-oriented environment, components might well include objects, many of which will be represented by packages (Ada) or modules (Modula-2). The component vector for an object may include counts of entities (types) which the object maintains, data elements of those entities, operations on them, and possibly imports. There are potential difficulties, however, where the specification is not in terms readily related to the design and implementation technology.

For size estimation the use of FPA-type metrics is, of necessity, technology-dependent. However, technology dependence is not concentrated solely in a LOC/FP ratio, but occurs also in the estimation model itself, in the division of modules into types with different estimators, in the relative weights given to estimates of different module types, and in the estimation formulae for each module type [VERN89].

Oversimplified Classification of Component Type Complexity Levels

The classification of component type instances into simple, average and complex has the merit of being simple but in practice is rather oversimplified and a component with "over 100 data elements is only given at most twice the points of a component containing 1 data element" [SYMO88]. This may not matter much for some system size estimates, where component sizes may average out, but may be important for other systems where the average component size is consistently different from the so-called average of Table 2.1 [VERN89]. This oversimplification is also important when estimating individual component sizes. The extension of the number of complexity levels of the component types from three as described in [ALBR83] to five or even six in some later models [DACS87] may partly overcome this criticism.

Choice of Weights

The weights given to the different component types were determined by Albrecht by "debate and trial" [ALBR83] and are meant to represent the relative value of the component to the user. In discussions with users, not all have agreed with Albrecht's weightings and some more objective measurement may be needed. Some of the weights give surprising effects at times. Symons [SYMO88] asks why an inquiry from a batch input/output system gained more than twice as many function points as the same inquiry provided on-line and why an interface to another system should be of more value to a user than any other input or output.

Complexity

FPA treats the system as if all inputs, outputs, and inquiries flow into or out of a black box process whose processing complexity (and size) is related to the number of data elements and logical file types that are referenced. This may give a guide to the complexity of some components but the complexity of others may not be adequately represented and that of yet others may be totally ignored. FPA appears to undercount systems that have fewer inputs and outputs using larger numbers of data elements and complex processing when compared with those that have a greater number of simpler inputs and outputs, and less complex processing [SYMO88]. An attempt is made in the system adjustment factors to overcome part of this problem; here the effects of interactive interfaces and complex processing on the system as a whole, together with other cost drivers, are used to modify the raw function point count.

Summation Problems

Because of the way the interface files are counted, where a single interface file is credited to two separate systems, the sum of the parts can be greater than the whole. If the system were developed as one (possibly) more complex system it would score fewer function points than if developed as two, (possibly) simpler systems but with similar functionality from the user point of view. Also, an integrated file is likely to score fewer function points than a collection of separate files.

(b) *Adjustment Factors*

The problems with the system adjustments relate to the restricted number of factors considered, the weights given to them, difficulties in measuring complexity, and the inherent subjectivity in applying the adjustments. These adjustment factors are the weakest part of the method [VERN87] with many of them relating to cost rather more than size. FPA system adjustment factors are discussed further in 3.2.3 while adjustment factors in general are discussed in 4.5.

Number of Factors

The number and type of system adjustment factors is unlikely to remain constant over time. Already Albrecht has changed them since his first introduction of FPA. The original version [ALBR79] used ten adjustment factors while the later version [ALBR84] includes fourteen. Some of the current factors appear to overlap and Symons [SYMO88] suggests that a more open-ended approach and some reshuffling of the current factors may be more useful. Some of the current adjustment factors seem rather old-fashioned, tending to be aimed at batch processing systems [VERN87].

Weights and Subjectivity of Adjustment Factors

The range of 0-5 used for the degree of influence of each of the factors is simple in concept but may not always be valid. Symons [SYMO88] gives an example of a factor that deals with multisite implementation and suggests that the system adjustment factors and their weightings should be re-examined. The system adjustment factors give small additive corrections with equal maximum weight (an unlikely situation in the real world) and, with a relatively small total effect, have a possible range of from ±35% adjustment. Given the subjectivity of the adjustment criteria, and a tendency of assessors towards means, the overall adjustment of the raw function points will probably rarely be more than 15% [VERN87]. Similar factors (cost drivers) in Boehm's COCOMO cost estimation model have widely varying weights and a much larger potential effect in total [BOEH81 p.118].

Complexity

As noted in (a) above, the raw function point count does not always measure the effect of processing complexity. FPA tries to overcome this obvious flaw with its system adjustment factors. Symons [SYMO88] considers that the way the internal complexity is dealt with is "rather inadequate and confused" and that systems studied by him with a high internal complexity do not seem to have their size adequately reflected by the FP method. The designers of the systems which Symons studied were of the same opinion.

(c) *General Criticisms*

Because the method is rather subjective (with guide-lines), and different installations are likely to use the method in somewhat different ways, it is unlikely that it will be possible to adequately compare the results and productivity figures between different installations. Low and Jeffery [LOW88] showed that function point estimates of size were lower for analysts experienced in both software development and in function

point estimation than for inexperienced analysts and that even among experienced analysts variations occur. They thus conclude that there is a need for organizational standards to be applied to the function point estimation process. Jeffery et al [JEFF88] express reservations with function points because of the subjectivity in the counting process.

Notwithstanding these criticisms, the use of function points does overcome some of the difficulties that have been found with LOC, they are available early and they are the only metric in common use that can make a claim of any kind to technology-independence.

(v) Approaches to Solving the Problems of Function Points

Symons [SYMO88] has suggested an updated version of this measure to solve some of the problems he identified in FPA. He calls this new version MarkII.

2.5.2 MarkII

MarkII is a composite metric introduced by Symons [SYMO88] in 1988. He suggested that all the logical transaction types making up an application should be approached in the same way, with the size of each transaction being calculated from the numbers of input data elements, output data elements and entities (replacing the older file concept) used by the transaction. Symons suggested that:

(i) interfaces between systems should be counted in exactly the same way as any other input or output

(ii) inquiries should be counted as any other input/output combination

(iii) the concept of logical file is impossible to define unambiguously, is not appropriate to a data base environment, and should be replaced by the (suitably defined) concept of an entity.

Symons doubts the validity of talking about value to the user and concentrates on development effort. In doing so he reduces the usefulness of MarkII for productivity studies involving different technologies. He believes that the work of McCabe

[MCCA76] can be used to give a complexity parameter to the size of processing. However, since this cannot be evaluated until after the system has been developed, he makes use of several of Jackson's [JACK75] general ideas relating the complexity of processing to the number of data structure components processed. This takes the form of a count of the number of entity types referenced by a transaction type. From this a new formula for calculating unadjusted FPs is obtained.

(i) MarkII FP Calculation

To obtain the raw function points (which Symons also calls the information processing size) the counts of input elements I_k, entities E_k and output elements O_k referenced in the kth transaction are summed over all transactions using the formula

$$\sum_k (0.44I_k + 1.67E_k + 0.38O_k)$$

in which the technology-dependent weights shown for the elements relate to the systems studied by Symons.

(ii) Calibration of MarkII

Symons proposed the above weights for the types of systems, "normative technologies" and environments he investigated but suggested that any other technology would have its own weights. Changes in technology would be shown by changes in the weights. Symons calibrated his MarkII measure against Albrecht 's function points using an analysis of effort required to develop input, output and processing (estimated by entities processed) respectively. However the Albrecht function point count for a system does not necessarily equal that provided by MarkII. The biggest differences were shown for systems gaining over 500 function points with FPA - the size up to which the calibration of MarkII was pegged to Albrecht function points. MarkII in the main appeared to give relatively higher function point counts for larger systems.

(iii) Problems with MarkII

Although MarkII does overcome some of the criticisms of FPA it unfortunately introduces some problems of its own.

By including files and interfaces as component types, FPA attempts to take account of the inherent size of the data model itself apart from its use in transaction processing. MarkII however, concentrates exclusively on processes and only includes files (entities) as and where referenced by a process with interfaces being treated as any other input and output (between applications). Because interfaces and files are not treated separately in MarkII they are de-emphasized. The problem with this is that the data model definitions, including files, may form distinct components or modules of a system in addition to, and quite apart from, file references in processes.

Symons applies weighting factors to FP elements (input elements, entities, output elements) based on the degree of difficulty of the development task for the particular technology being used. In one sense, there is nothing wrong with this; it certainly appears to be more reasonable than 'function value delivered to the user' and may well, in some cases, give a better size estimate for an application, in a specific technology, than that given by FPA. The calibration of MarkII FP elements with effort, however, destroys what is possibly the major strength of FPA, its use in productivity studies. For this purpose it would be better to calibrate the MarkII elements with a size-based metric, not an effort-based one because the former is somewhat closer to the inherent 'information processing size'. The size-based measure can then be calibrated back to function points for productivity studies.

2.5.3 BANG

DeMarco [DEMA82, DEMA84] suggested an approach to the measurement of system size that he called System BANG. This object-based measure is calculated from Structured Analysis specifications [DEMA79, GANE79]. System BANG "represents the weight of function to be delivered" [DEMA84 p.21]. The system specifications are developed down to functional primitives. A functional primitive is described as "a trivial piece" ... "too small to justify further partitioning" [DEMA84 p.18]. Each functional primitive is given an empirical complexity correction factor. This factor depends on the class of primitive function. DeMarco defined 16 classes (a beginning set) of primitives although he suggested that the correction factor value for

some may be less likely to remain invariant than for others. Table 2.3 shows the classes of primitives and correction factors that were included in this beginning set.

Class of Primitive	Correction Factor
Separation	0.60
Amalgamation	0.60
Switching	0.30
Simple Update	0.50
Storage Management	1.00
Edit	0.80
Coherency	1.00
Text Manipulation	1.00
Synchronization	1.50
Formatting	1.00
Display	1.80
Summarizing	1.00
Arithmetic	0.70
Initiation	1.00
Computation	2.00
Device Management	2.50

Table 2.3 Functional Primitives and their Correction Factors

The number of input and output elements at the boundary of each functional primitive is summed and used to adjust the value of the functional primitive. DeMarco provided a table of weighted functional primitive increments for this adjustment factor and stated that the values were based on "Halstead's volume/vocabulary relationship" [DEMA84 p.18, HALS77] (see section 2.5.4). The resulting values are summed to give total System BANG. The formulation is thus:

$$\text{System BANG} = \sum_k (.5n_k \log_2(n_k))C_k$$

where

C_k = weight of kth component (functional primitive)

n_k = number of input and output elements at the boundary of the kth component

$.5n_k \log_2(n_k)$ is the adjustment factor formula.

DeMarco found the measure was not installation-independent and suggested that "the complexity weighting factors are, unfortunately, environment dependent" and that two of them, device management and computation, "you correct with your own estimate based on type of computation or type of device" [DEMA82]. He also suggested that "you will need to develop your own set of weightings and perhaps some of your own new classes" [DEMA82].

DeMarco [DEMA82] also suggested a second System BANG measure. If the system is data-strong, i.e. one with a significant data base, then the BANG can be based on the count of objects in the data base. Each object count is corrected for the number of relationships at the object boundary and the corrected objects are then summed. For systems that are both data strong and function strong he suggested that the system should be divided, and two sets of BANG metrics should be used. The two predictors should not be combined and the project should be treated as if it were actually two projects. He did not believe that there was any satisfactory way of combining Function BANG and Data BANG in the general case but that an individual installation may be able to get a procedure for relative scaling so that the two could be added together. This implies that DeMarco had a metric vector view of the two counts. If a satisfactory scaling procedure could be found the two measures would be converted into a single composite metric.

System BANG has some appeal as it is developed from specifications but the functional primitives are at such a low level that the information from it would not be available until the end of the design phase. There has been no other work published regarding the use of this metric.

System BANG (at least from the published material) appears to be a general approach to a sizing method, that different users can customize in different ways, rather than a precisely defined method that they can follow. It is too subjective and the work involved for a DP department to build up its own weightings and classes of primitives would tend to prohibit wide use in its present form. This would also result in the measure becoming too installation-dependent. The System BANG approach is a very promising one if it can be built into an automated specification tool so that it can be calculated automatically and objectively as specifications are developed, for example, in conjunction with one of the Structured Systems Analysis and Design CASE tools that have recently been developed in Europe and the United States. Indeed, until it is automated, it is unlikely that it will be used at all widely.

2.5.4 Software Science

Software science [HALS77] is a family of composite metrics. Halstead used operator and operand tokens as the fundamental counts in these measures. He tried to go beyond LOC, with software science, by combining both the data and the

functional aspects of code from a program's vocabulary to calculate both program length and 'volume'. A program, procedure or function was considered to be a collection of tokens that could be classified as either operators or operands. This is based on the fact that all programs can be reduced to a sequence of machine language instructions each of which contains an operator and a number of operand addresses [SHEN83 p. 155].

Halstead defined the following basic measures:

n_1 = number of distinct operator tokens in the program
n_2 = number of distinct operand tokens in the program
N_1 = total number of occurrences of operators
N_2 = total number of occurrences of operands.

The **vocabulary** of a program is

$$n = n_1 + n_2$$

and the **length** of a well-structured program was a function of the number of operators and operands

$$N = N_1 + N_2$$

where N is closely related to the traditional LOC measure of program length. For machine language programs where each line consists of one operator and one operand, N = 2 x LOC [SHEN83 p.156]. Shen also states that, "like many other software metrics, N may not be a precise equality for a specific program yet may be considered valid in a statistical sense".

If a program is made up of k procedures and functions its length is then:

$$\Sigma_k (N_k)$$

Program volume defined as the volume of an implementation of an algorithm in bits is

$$V = N \log_2 n.$$

Programming practices such as the redundant usage of operands or the failure to use higher level control constructs will tend to increase the volume [SHEN83 p.156]. Tokens that represent data, variables or labels are defined as operands while those tokens that specify an action are considered as operators. Punctuation marks, arithmetic symbols, command names, function names, special symbols like brackets and keywords are categorized as operators. Length and volume are two measures of program size.

If N_1 and N_2 are not available the length equation

$$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

is claimed to yield a good approximation to the length N. The length of the program can be estimated using this equation provided that the distinct operators and operands can be counted before the program is written. From the length metric effort and volume measures could be estimated. N may be converted to an estimate of source LOC via the relationship

$$S = N/C$$

where the constant C is language-dependent. For FORTRAN C is thought to be about 7 [CONT86 p.41].

(i) Criticisms of Software Science

There have been a number of criticisms of software science [LIST82, SHEN83, LEVI86]. Those criticisms that relate to software size are discussed below. They fall into five categories:

(i) derivation of the formulae
(ii) experimental work
(iii) operator and operand definition and counts
(iv) length equation
(v) volume equation

a) *Derivation of Formulae*

Software science as originally defined dealt with algorithms, consisting of "operators and operands and nothing else" [HALS77 p.8], not programs. Although rigorous algebraic derivations for the formulae are given in [HALS77] several assumptions are made "for which there would seem to be no theoretical justification" [SHEN83 p.157]. The assumptions that operators and operands alternate seems reasonable but this would imply that $N_1 \approx N_2$ which has not been observed in general [SHEN83 p.158]. Lister [LIST82 p.70] suggests that the "foundations of software science are perilously weak" and that "it needs a clearer definition of its assumptions, goals and domain of application" as well as "a methodology which is seen to be rigorous". If it develops in these ways it may provide "useful and worthwhile results".

b) *Experimental Work*

In many of the reported experiments the sample size is too small for great significance to be attached to the results [LIST82 p.67]. Many of Halstead's conclusions were based on sample sizes less than 10. The programs involved were small. It is probably not possible to generalize results from such programs to large, multi-module industrial programs. Many of the experiments involved single subjects and the subjects, even when there were several, were college students. The results may not be applicable to professional programmers [SHEN83 p.158].

c) *Operator and Operand Definition and Counts*

Just as there are problems with counting LOC, there are variations in classifying operators and operands in software science. It is important that the counting scheme be clearly defined and consistent across experiments [LIST82, p.67]. There is no general agreement amongst researchers on the most meaningful way to classify and count these tokens [SHEN83]. The consequences of this are serious as it was shown by Elshoff [ELSH78] that small variations in a counting scheme can affect certain measures by as much as 50 percent. The rules used for operator/operand classification are language-dependent and frequently ambiguities occur in the counting of the unique operands [CONT86 p.38]. The conventions used for counting operators and operands should be applicable to programs written in any of a wide class of languages [LIST82 p.67]. The counting scheme initially described by Halstead [HALS77] was for FORTRAN programs. The scheme is more difficult to apply to more modern languages and to less primitive control constructs [LIST82, p.67]. For languages like LISP, the actual existence of operators and operands has been questioned altogether [LASS81]. The nonprocedural parts of 4GLs consist of

mainly of operands with most operators being implicitly supplied by the positioning of the operand.

Later work [SHEN83 p.157] has included declarations and input/output statements which the early counting conventions ignored. Statement labels were originally considered to be operators but recent research tends now to classify labels as operands [CONT86 p.37]. There are also problems with counting symbols that are used with different meanings in the same program. Should they be counted separately for each meaning? Levitin [LEVT86 p.316] suggests that the consideration of tokens like '( )', 'BEGIN-END', and 'IF-THEN-ELSE' as single operators has major consequences for the length metric and that the pitfalls of the operator/operand dichotomy are especially annoying since both length and volume depend on the sums of the operator and operand counts, which makes a distinction between them irrelevant. He suggests that we would be better to just count tokens.

d) *Length Equation*

There is no rigorous mathematical derivation for the length equation [FITZ78, p.6]. Card [CARD87 p.32] states that there is a mathematical dependency in the length equation that explains why this equation has appeared to have deceptively strong empirical support and that the failure of the length equation threatens the foundation of software science. Lister believes that, although the empirical evidence has shown that the length equation works and initially appears impressive, on closer examination this evidence is not so conclusive as it first appears [LIST82, p.67]. Lister believes that the length equation is not well established and that it may not hold for programs written in structured languages unless a counter-intuitive counting scheme is adopted [LIST82 p.67]. All the early empirical evidence supplied was from programs written in FORTRAN and PL/1 or from small Pascal programs written by students (with presumably a high proportion of impurities). Lister [p.68] cites nine large professionally-written Pascal programs that show large discrepancies between values of N and ^N. These discrepancies were so large as to provoke further analysis. Examination showed that for these large programs n_1 was consistently less than n_2 whereas for small programs n_1 and n_2 were comparable. This observation suggested that ^N falls short of N because the contribution of n_1 is too small. This is caused by the programs being large enough to contain most of the built-in operators so that the growth of n_1 is constrained by the number of user-defined procedures and functions. Analysis of these large programs showed that there were not enough user-defined operators to give a value of n_1 large enough to satisfy the length equation. The situation was different with the FORTRAN

programs, which provided the early empirical support, as nearly all transfer of control was by jumping to a label. These labels made a large contribution to n_1. Lister altered his counting scheme to count each occurrence of a control construct as a distinct operator to be counted each time it was included. He suggests that such a counting scheme is most unappealing but did give rather better results. The early PL/1 programs contained a significant number of GOTOs despite the structured nature of the language and this increased significantly the size of n_1. Lister also suggests that there was no more evidence to support

$$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

than there was to support a number of other arbitrary functions such as

$$n^2 \text{ or } 10n_2 \text{ or } n_1+n_2 \text{ or } n_1^2+n_2^2$$

as estimators for N in his programs. He notes that $10n_2$ gives a better fit for the 31 programs used by Shen [SHEN79] but does not propose it as a new estimator [p.68]. Lister concludes [p.70] that the length equation may only hold for languages with primitive control constructs.

Basili [BASI83 p.658] reported that in his experimental work ^N significantly overestimated N for modules below the median size and underestimated those above while Card and Agresti [CARD87 p.30] also report that N consistently underestimates large modules. Wang [WANG85] found that "the length equation is not suitable for large Pascal programs".

e) *Volume*

Levitin [LEVT86 p.317] suggests there are special problems with the volume metric and that there is even some uncertainty regarding its definition; on one hand it is measured in bits but on the other hand it can take non-integer values (with a range that is not continuous either). The values can be rounded up but this has not been done consistently in the general literature or by Halstead [HALS77] himself in the original monograph on software science. Levitin also shows that volume is not additive and that the volume, in bits, may not always be adequate to encode a program fragment. He cites an example where this is caused by the treatment of bracket pair as a single operator.

(ii) Approaches to Solving the Problems of Software Science

Software Science counting schemes have been identified for many common languages [LEVI86] however these counting schemes have not always shown consistency. Some sets of counting rules have been published, for example, for Pascal in [CONT86 p.38].

Albrecht and Gaffney [ALBR83 p.640] demonstrated that the software science formulas originally developed for small algorithms could be applied to large application programs in Pl/1 and COBOL. They also relate software science measures to function points and source LOC, and suggest that both the work hours and application size in source LOC are strong functions of function points and input/output data item counts (n^*_2). They use n^*_2 to mean the sum of the overall external inputs and outputs of the application program.

Albrecht and Gaffney state [p.640] that n_1 (number of unique operators) need not be known to estimate the number of tokens N, or the number of instructions for a single address machine. "An 'average' figure for n_1 (and thus for $n_1 \log_2 n_1$) can be employed or the factor $n_1 \log_2 n_1$ can be omitted, inducing some degree of error". They suggest that one could take a number of approaches to estimating program (code) size. The "data label vocabulary size (n_2)" could be estimated, or n^*_2, the number of conceptually unique inputs and outputs can be used as a surrogate for n_2. They state that n^*_2 should be easy to determine early in the design cycle from the itemization of external inputs and outputs found in a complete requirements definition. Albrecht and Gaffney found that a sample correlation, for 29 APL programs, between N and $n^*_2 \log_2 n^*_2$ was 0.918 and between N and $n_2 \log_2 n_2$ was 0.988 and relate their results to function points, to source LOC and work hours. Their measures based on I/O count showed slightly better, but not significantly better, statistics than those based on function points.

(iii) Other Work Using Software Science

In System BANG DeMarco [DEMA82, DEMA84] summed the number of input and output items at the boundary of each functional primitive and then used this sum to adjust the value of the functional primitive. He provided a table of weighted functional primitive increments for this adjustment (Table 2.3) and stated that the values were based on Halstead's volume/vocabulary relationship (2.5.4) [HALS77].

Albrecht and Gaffney [ALBR83] published work relating function points to software science and suggested that the function point software estimation procedure appears to have strong theoretical support based on Halstead's [HALS77] software science formulas. They suggest that their work can provide an early bridge between function points, software science and source LOC.

2.6 PRIMITIVE, COMMON, COMPOSITE AND DERIVED METRICS

In this discussion of software metrics it is important to note that some metrics are estimators while others are estimated. At one stage in software development a metric may be *estimated* from an *estimator* but at a later stage of development the metric of interest may be counted directly; for example LOC may be estimated from a count of objects occurring in the specifications but at completion of the software development lines may be counted directly. Because other metrics like function points and BANG have no real existence in the sense that there is no discrete thing one can point to and say 'That is a function point', they can never be counted directly but only ever estimated or calculated. Metrics that can be counted at some stage in development can be termed **primitive metrics** while those that cannot be counted directly but are estimated or derived from primitive metrics can be termed complex or **derived metrics.**

Primitive refers to a measure that is

(a) conceptually or intuitively simple, though its complete and unambiguous definition may not be easy
(b) can be counted directly at some stage during development.

Primitive metrics hence include counts of objects of various kinds, statements, LOC, tokens, characters.

Common refers to a measure that is widely applicable to many development methods and to many languages. Primitive measures tend to be also common.

Composite refers to measures which are functions of more than one other measure and/or count, for example a function of the number of data elements, the number of

entities and the complexity of a set of associated procedural fragments in a particular component. FPA-like measures are composite.

The term derived has the additional connotation of being related through some explicit or implicit calibration process to some other estimate or measure. For example, an FPA-like estimate (as established in 4.3.7) may be derived from an estimate in LOC by determining the ratio of LOC per function point.

Composite measures tend also to be derived because they need a basis for combining different measures and object type counts, unless this is to be done arbitrarily, or 'by debate and trial' as in Albrecht's FPA [ALBR79]. Thus, the main types of measures are primitive and composite derived, referred to below as primitive and composite, respectively. Some primitive measures, such as LOC, are common, as are some composite measures such as function points.

Figure 2.2 shows diagrammatically the relationships of these measures and where current software size metrics fit into this scheme.

2.6.1 Uses of Primitive and Composite Size Measures

The uses of primitive and composite measures tend to be different. While primitive measures can potentially be used at the requirements or specification stages of development, for example by measuring the textual bulk of a specification, composite measures in terms of counts of specification objects tend to be more meaningful and useful at this stage. However, the use of formal automated specifications in both CASE tools and executable specification systems is likely to stimulate the search for suitable primitive specification measures (see 2.7). Primitive measures tend to be applicable to source code of most types. However, the trend to non-procedural, form-filling development methods in CASE tools and 4GLs, makes the application of some primitive measures, such as LOC, more difficult in such cases.

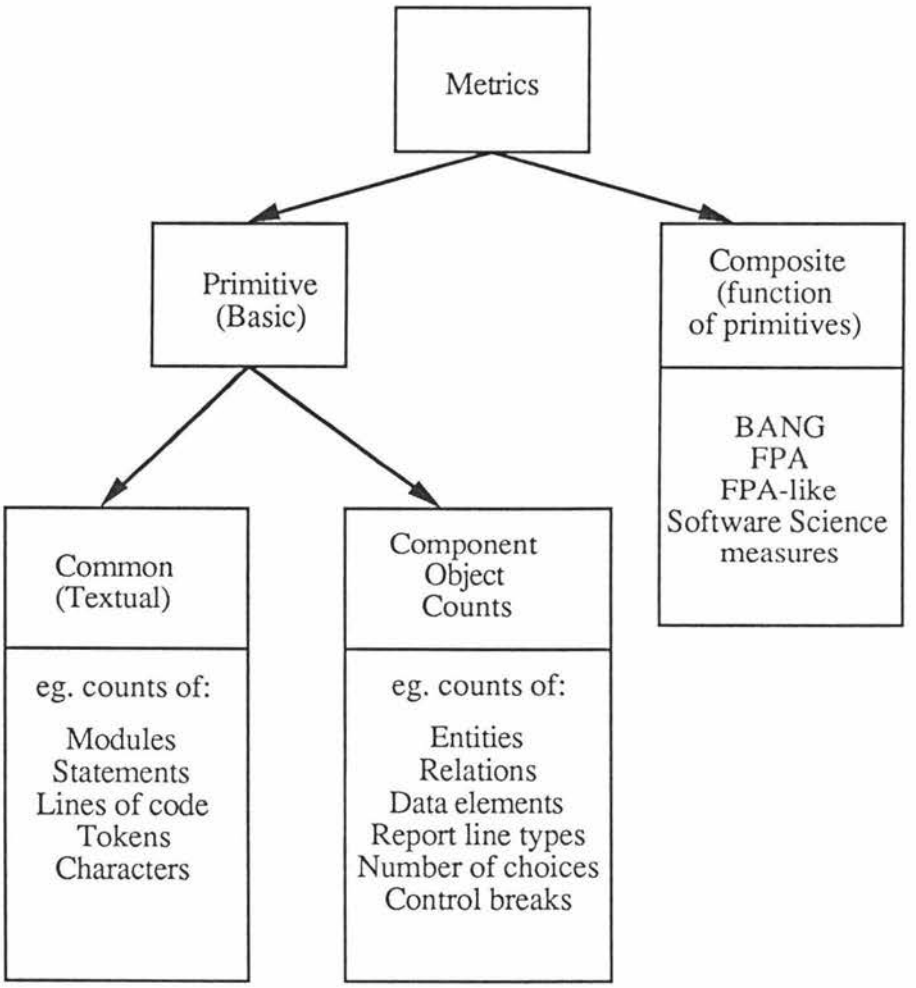


Figure 2.2 Relationships Between Metrics

What is being measured or estimated by the metrics may vary. A measure may be used to obtain the size of a specification or to get the 'intrinsic' size or size of a job to be done. It may be a measure or estimate of a partial size of a system or it may include adjustment factors in an attempt to estimate the size of the system as a whole. When a metric is used may also be important. A metric may be used to measure or estimate the partial size or entire size of a specification, design, or implementation and different metrics may well be required for each of these sizes. What the measure is being used for may also be important; for example it may be for early size estimation or may be for data collection at the end of the development.

2.7 NEED FOR FUTURE MEASURES

The increasing use of CASE tools has created a need for size metrics useful for developers using these tools. Metrics need to be able to relate to the LOC and FP measures that have been used for so many years, to allow comparisons against projects developed with conventional tools for productivity measurements. Metrics must also be available for comparative productivity studies between projects developed with different CASE tools. It is possible that there may increasingly be a need to move towards the concept of a metric vector to satisfy these different requirements in a software size metric. Measures of size, obtainable from specifications, which can be calculated automatically from entries in the system dictionary, will increasingly be required by users of CASE workbenches and tools. These users' major interest will not be how many LOC will be generated by the CASE system employed; their interest will rather be in metrics able to measure the size of the job to be done, in a form useful for scheduling purposes, and perhaps also for measures that can also be used for comparative productivity studies.

2.8 SUMMARY AND CONCLUSIONS

This review of previous work has classified software size metrics into four groups, textual metrics, object counts, metric vectors and composite metrics. An additional classification of primitive and derived metrics was also briefly discussed. This is a more comprehensive classification and review than any previous one the author is aware of.

There are no software size metrics that are completely satisfactory, even for a single well-defined purpose. The unsatisfactory features of each metric have been analysed as well as approaches to overcoming them. The usefulness of a metric is dependent on the purpose for which it is being used. A metric that is appropriate in one situation may be quite inappropriate in another. No one metric discussed was seen to be useful in all situations. For many purposes a single LOC measure may be appropriate but for productivity purposes function points may be used. In this context function points are a more meaningful metric than LOC measurements which are too language-dependent. However, for input into most current software cost estimation models LOC metrics are essential. LOC is unlikely to be replaced in the short or medium term and any emerging metrics will need to be capable of being converted to LOC for some time to come.

With the increasing use of modern software development technologies like CASE tools, other measures of system size, based on counts of a variety of objects or tokens occurring in the specifications or software, may become more important. System and Data BANG introduce some important principles in this respect and metrics like them, based on object counts, warrant further research.

Because software size is often determined by counting a number of different objects there is no one single common count from which it can generally be determined. This consideration leads to the important concept of a metric vector including several counts or measures of different objects or aspects of a system or component. This concept of a metric vector, which counts more than one object within specifications or software, is identified as likely to be of increasing use as an approach to software size metrics.

Single valued composite or derived metrics which attempt to combine several of the basic or primitive counts into a single metric are required for comparative studies. Such metrics are essentially functions of metric vectors. FPA and its derivatives are a useful step in this direction.

FPA and MarkII are examined in detail, including a detailed analysis of the criticisms that have been levelled at them. It is concluded that neither FPA nor MarkII function points are technology-independent though both are much less technology-dependent than LOC. The treatment of adjustment factors in FPA is seen to be particularly unsatisfactory for sizing purposes.

There is too much controversy about the usefulness of software science. There has been a good deal of research on the various software science measures with much of it giving very mixed results. Because of this, and because software science metrics are not used widely in practice, software science *per se* is not considered further in the context of this thesis.

The metrics identified as being of greatest importance to this thesis include token counts, statement counts, LOC, various object counts, metric vectors and FPA-like composite metrics.

CHAPTER 3

CURRENT APPROACHES TO SOFTWARE SIZE ESTIMATION

As discussed in Chapter 1, there have been, and still are, many problems with the accurate estimation of software size prior to the implementation of an application. Because of the problems a number of different approaches to software size estimation have been developed. Many of these produce size estimates in LOC for the required language, although some models allow for estimation in more than one metric (for example, SSM [BOZO86]).

A recent report for the United States Air Force [DACS87] describes a number of software size estimation models and evaluates six computerized models against a known (completed) real time project. The results are five over-estimations ranging from 27% to 400+% and one under-estimation (28%). The overestimates described in this report may well have been caused by the post-implementation nature of the study. With a completed system there is less likelihood of overlooking parts of it, whereas inadvertent omissions are frequently a problem when estimating systems before development and are therefore likely to be allowed for implicitly in some models.

Boehm [BOEH81 p.320] discusses common reasons for inaccurate early size estimation - usually under-estimation. These include:

> "people are basically optimistic and desire to please"
> "people tend to have incomplete recall of previous experience"
> "people are generally not familiar with the entire software job".

The earliest unit used to measure software size, and the most familiar, is the LOC. Historically, managers relied on expert judgement, based on experience and on analogy with projects of similar characteristics, to establish the ultimate size of the project [CONT86 p.214]. The interest in LOC as the metric for software size

estimation is a result of much real-time and embedded software production in the United States for the Department of Defense and NASA, and by some large companies (for example, TRW, Rockwell, Lockheed, Boeing) which are involved in the development of very large software (and hardware) systems. These organizations use cost and schedule estimation packages that require LOC input (for example COCOMO [BOEH81, BOEH84], PRICE S [FREI79] and SLIM [PUTN78, PUTN79, QSM87]). The use of costing and scheduling models by these large organizations has resulted in relatively more research into the estimation of size (and cost) of real-time and embedded systems. In comparison, the estimation of size of business applications has been relatively neglected except for the major work done by Albrecht [ALBR79, ALBR83] of IBM. In fact Dr. Donald Reifer, the developer of ASSET-R [REIF87, REIF87a] stated that there was great difficulty in acquiring business data processing sizing and project data [REIF87b]. This relative lack of data and small amount of research in the business applications area has been brought about by the comparatively small size of most business data processing departments. Their software developments, when compared with developments like the United States Defense Department's Strategic Defense Initiative (SDI), are also relatively small.

Though a sizeable amount of work has been done, mainly within large organizations developing real-time systems, there has been relatively little published research in the software size estimation area probably owing to business considerations.

Though LOC are still the most commonly used metric, function points are increasing in popularity as an alternative size metric for some purposes. Not only is there no current agreed metric for software size measurement (or estimation), there is also no one generally accepted approach to obtaining software size estimates. None of the current approaches has been shown to be entirely satisfactory and nearly all have been criticized for inadequacy in some way or other [ALBR79, LIST82, RUDO82, JONE86, VERN87, SYMO88, VERN89]. Almost all current approaches to software size estimation involve some degree of subjectivity or expert judgement. While some approaches rely solely on expert judgement, others provide a framework of procedures aimed at assisting the expert(s) to produce more accurate estimates. Some of the earlier approaches to estimating software size, such as expert judgement and PERT techniques, have been incorporated into parts of newer models.

This chapter presents a review of current software size estimation methods. A new classification of approaches to software size estimation, developed by the author, is introduced in section 3.1 and discussed in detail in section 3.2. The only published references to some of the material, available to the author, is the DACS Report [DACS87] and one of the automated models described in that report, Software Sizing Analyser (SSA), is only available to United States Government agencies.

3.1 CLASSIFICATION OF SIZE ESTIMATION APPROACHES

Size estimation approaches can be classified in a number of different ways. Four possible classifications, structural, general (DACS87), temporal and metric-based, are identified and discussed in this chapter.

1) A new classification of software size estimation approaches, developed by the author, is a **structural classification**. This classification is based on the type of structural partitioning of a specification (or design) that must be completed before the method can be used. Using this classification estimation approaches have been subdivided initially into two major classes:

   a) those that estimate the total size of the proposed system as a single entity and

   b) those that require the specifications to be first divided into components, with size estimation being done at the component level. The total estimated size, for this class, is the sum of component sizes.

   The second of these major sub-classes is then subdivided further. The first sub-class makes no distinction between components while the second groups components into several different component types. Both the resulting sub-classes may be further partitioned by a consideration of their approach to estimating component sizes; a component may be estimated by type, without consideration of its individual characteristics (i.e. an average size is used for the component types) or, on the other hand, a component's size may be individually estimated (within its component type if there are several types) from (possibly) some set of object counts.

The structural classification described above is shown in Figure 3.1 and software size estimation approaches are discussed within this structural classification in section 3.1.1.

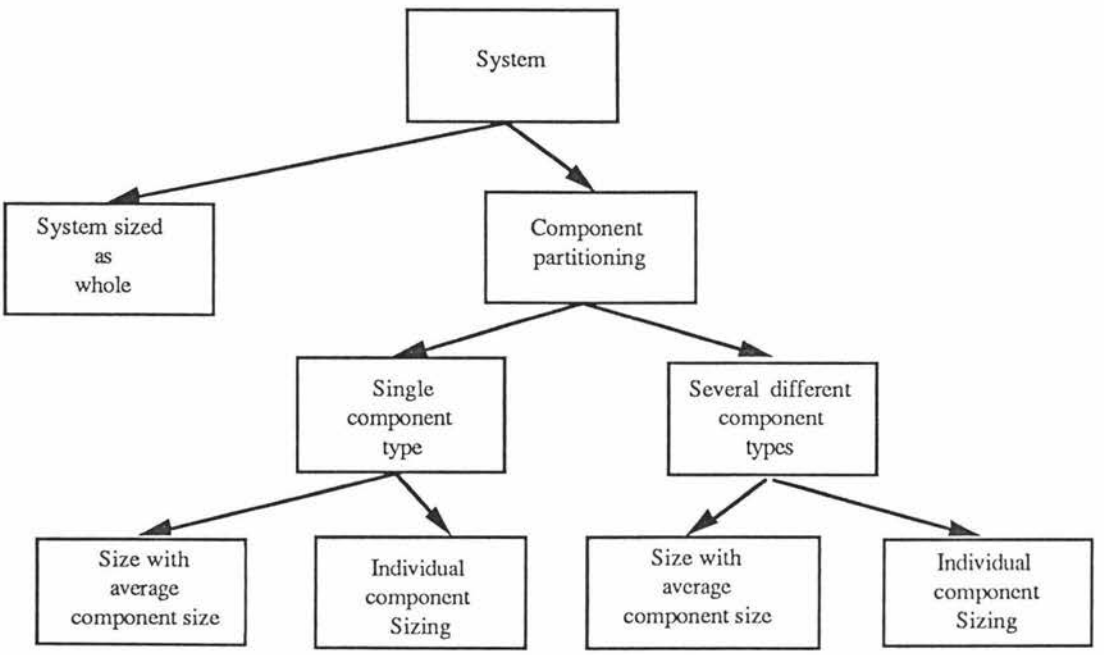


Figure 3.1 Structural Classification of Software Size Estimation Methods

2) The classification used in the DACS Report, is based on "categories that illustrate general approaches to sizing" [DACS87]. The following classes are identified:

sizing by analogy
size-in-size-out
function point analysis
comparison of project attributes
linguistic approach
other.

However, some estimation methods incorporate more than one approach and would thus be included in more than one of these categories. For example, QSM Size Planner [PUTN87], which uses three different approaches and can combine the three weighted outputs into a single size estimate, is discussed within three classes. This classification scheme is discussed in section 3.3 and is applied to a number of additional approaches that were not included in the DACS report.

3) A further alternative classification, developed by the author, and useful for rather different purposes, is a **temporal** classification. This classification is based not on the general approach used, but on a consideration of what parts of the specification or design must be completed before estimation can begin. Using this set of criteria, the following four classes are then identified:

   feasibility
   requirements
   preliminary design
   detailed design.

   There may, however, be variants to these classes depending on the software lifecycle model[1] adopted. Of course, all software size estimation methods tend to give better results as the development process proceeds and gaps in the required information are filled in.

4) An attempt was also made to classify the different software size estimation approaches on the primary metric used but this proved to be unsuccessful. Because some approaches give results in more than one metric, and some are able to produce results in any metric specified by the user, this classification had so many overlaps that it was abandoned.

None of the classifications discussed above is considered to be entirely satisfactory. The structural classification, however, appears to result in fewer problems. Therefore it has been adopted as the preferred classification and is used to provide the structure for this section. For comparative purposes both the DACS classification and the temporal classification are discussed in more detail later. Section 3.3 reviews

[1]This may of course include incremental development, prototyping, etc.

software size estimation methods from the alternative viewpoint of the DACS classification and section 3.4 reviews methods within the temporal classification.

3.2 STRUCTURAL CLASSIFICATION OF SIZE ESTIMATION APPROACHES

Major size estimation approaches are identified and discussed within the structural classification in sections 3.2.1 - 3.2.2. Figure 3.2 shows diagrammatically the relationships of the approaches discussed to each other, and their positions within the classification structure.

The first division in this classification partitions the approaches into those that estimate the system as a single entity as opposed to those in which the specifications must be sub-divided into components before estimation can be done. There are some methods that could fit into either of these categories. CEIS (Computer Economics Inc. Size Estimator), a recently developed commercial package, is one such method [DACS87 p.2-35]. This approach "allows the size of a task (program or project) to be estimated by comparing attributes of the new task to three reference tasks of known size". Although the size range of a task, or whether a whole system could be estimated as a single task with this package, is not specified there seems to be no reason why this should not be possible if three reference systems of known size are available. The CEIS approach however has been included in section 3.2.2 which deals with approaches requiring specifications to be partitioned into components because it is more likely to be applied to subsystems at some convenient level.

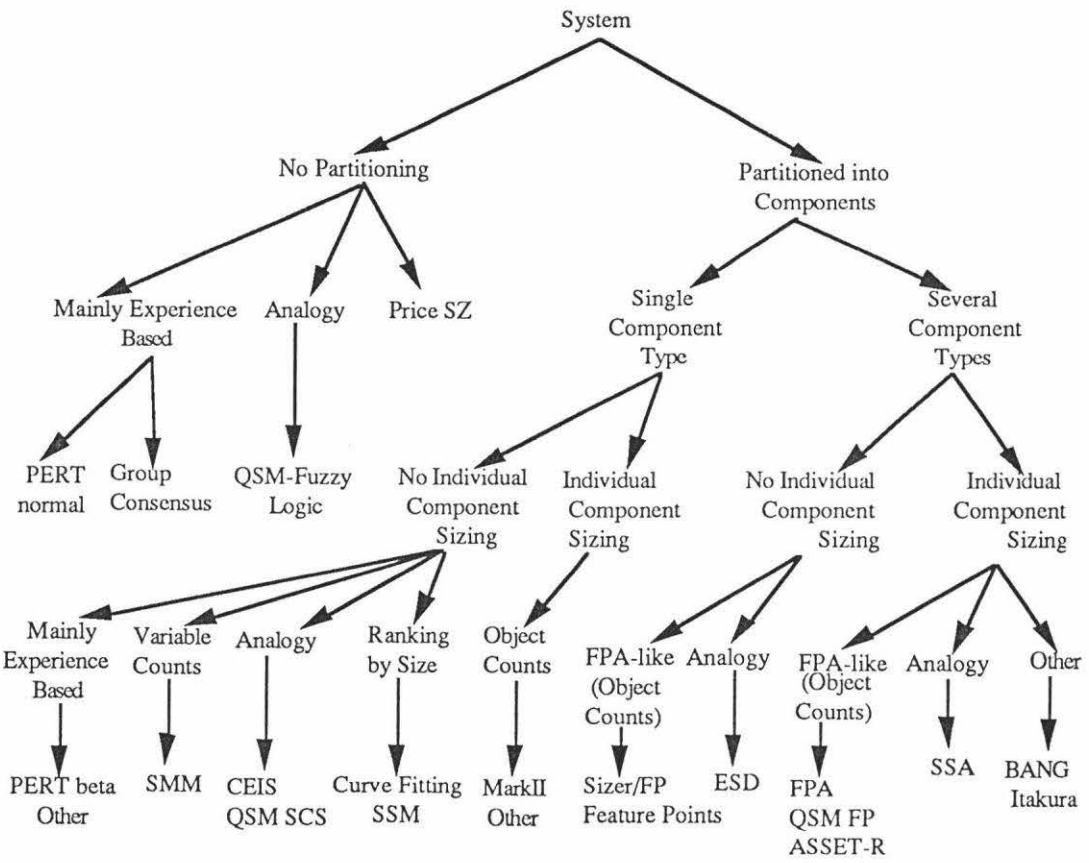


Figure 3.2 Structural Classification of Specific Software Size Estimation Methods

3.2.1 No Partitioning of Specifications

The estimation methods discussed in this section do not require the software specifications to be partitioned into components, modules, programs etc. before size estimation can be done. The following approaches are included in this class and are discussed below:

Experience based estimation

Group consensus including standard Delphi and Wideband Delphi

PERT sizing based on a normal distribution

Analogy

QSM Size Planner - Fuzzy Logic

Other methods

Price SZ

The PERT technique can be used at either the system level or the module level [PUTN79] and is thus also included in section 3.2.2.

(i) Experience-Based Estimation

Several experience-based approaches have been described in the literature. They are mainly aimed at assisting in the production of LOC size estimates. When they are used the target language or languages must be known beforehand and, without a thorough understanding of the specifications and familiarity with the specific type of application, it is unlikely that good estimates will result. Experience-based methods require initial estimates of approximate size values. The final size estimate is a refinement of the broader, inexact estimates provided by the input. These techniques may be used for size estimation (when size is to be input into a costing model), however they have also been used for the production of cost estimates and it is in this context that they are discussed in [BOEH81]. The expert judgement approach, he suggests, may use just one expert but the result will only be as good as that expert, and will be as biased, optimistic or pessimistic, as that expert is [p.333].

Group consensus, PERT and Delphi methods can be used for size estimation at both system level and module level. However, group consensus, the Delphi methods, and the PERT sizing method using a normal distribution are usually applied at the system level and are therefore discussed in detail in this section.

(a) Group Consensus

A variation of the expert judgement approach is a group consensus technique. This involves the estimation of size by a group of experts. The final size estimate is the mean or median of the individual estimates. Boehm [BOEH81 p.333] describes the method as being "quick but subject to adverse bias by extreme estimates". A suggested variation is to hold a group meeting in which the experts discuss their individual estimates, with the aim of getting them to converge on, or agree to, a single

estimate. Boehm suggests that this method may filter out some uninformed estimates but that group members may be influenced by the more glib or assertive members, or that the group may be overly influenced by politics or authority figures. This was certainly the case when the following experiment, a variation of this technique, was tried by the writer in 1985. A group of fifteen graduate students were the 'experts'. Most of these students did not have commercial data processing experience but they were all familiar with projects of a similar size to the system under discussion. Two of the members of the group were very assertive. The rest of the group were very disinclined to argue with them. The result was that the estimate, instead of being a consensus of fifteen, was made in reality by only two members of the group.

Boehm suggests that some of the drawbacks of this approach may be overcome by using a **Standard Delphi technique** as described by Helmer in 1966 [HELM66] and used as an expert consensus method in a number of different situations. Here the experts anonymously fill in their estimates on a piece of paper, a co-ordinator prepares a summary of the estimates with the rationales behind the estimates, distributes these to all participants, and then asks experts to give another estimate. Forms are filled out again and iterations continue until agreement is reached. No group discussion occurs.

Because this method did not provide a sufficiently "broad communication bandwidth" Farquahar and Boehm [BOEH81 p.335] in the early 1970's formulated "an alternative method called the **Wideband Delphi technique** which has subsequently been used in a number of studies and cost estimation activities". The standard Delphi differs from the Wideband in that group discussion does take place in the Wideband. "Group discussion offers considerations for estimating size of the development effort that may be otherwise overlooked. Generally discussion will filter out extreme estimates" [DACS87]. Boehm suggests that Wideband Delphi "has been highly successful in combining the free discussion advantages of the group meeting technique and the advantages of anonymous estimation of the Standard Delphi technique" [BOEH81 p.335]. In this variation, illustrated in Figure 3.3, an initial meeting of experts is called where estimation issues are discussed with the co-ordinator and each other. Forms are then filled out anonymously and a summary of these estimates are distributed to the members of the group. Another meeting of the experts is then called. They discuss points where the estimates varied widely and

then later fill in forms with revised estimates. Iterations continue for as many rounds as are appropriate.

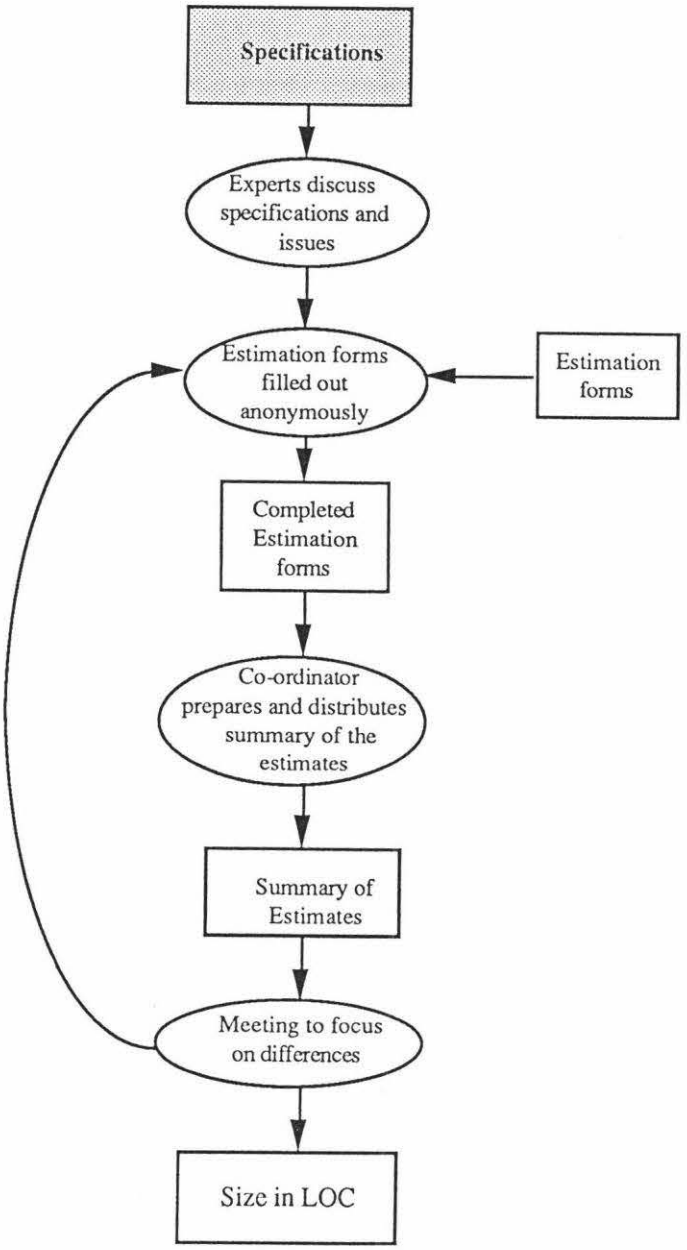


Figure 3.3 Wideband Delphi Technique

A variation on this method was tried in 1986 by the writer with a group of sixteen graduate students. These 'experts' were split into four groups of four. Each group produced an estimate of the product size. The chairman produced a summary of the

estimates and distributed them anonymously. The groups re-estimated and the results were again collected. Only one group, that which had provided the lowest estimate, was prepared to alter its initial estimate. The other three groups, perhaps with the confidence supplied by their peers, refused to change their estimates in any way. No agreement was possible, with all groups being quite adamant that they were correct. This resulted in three of the groups having significant underestimates (by 400% in the worst case), while the group that had made the highest estimate had only very slightly underestimated.

(b) PERT Sizing Based on the Normal Distribution

This is the simplest of the PERT approaches and involves the estimation of two sizes for the completed product by experts familiar with the proposed project specifications:

a = the lowest possible size of the software
b = the highest possible size of the software.

The expected size of the software is

$$E = \frac{a + b}{2}$$

and the standard deviation of the estimate is

$$\sigma = \frac{b - a}{6}.$$

These formulae are based on the assumption of a normal distribution of sizes between the two extremes a and b; a and b are understood to represent three standard deviation limits on the probability distribution of the actual software size. For a normal probability distribution this means that the actual software size would lie between a and b 99.7% of the time. Boehm [BOEH81 p.319] suggests that there is a problem with this method in that if the maximum size given for b corresponds to the maximum amount of code that will fit in the machine, then the result is unlikely to be accurate. He gives an example to support this criticism. PERT techniques have also been incorporated into several automated models including SSM [BOZO86] and Putnam's SLIM costing model [PUTN78, PUTN79, QSM87].

(ii) Analogy

QSM Size Planner - Fuzzy Logic

QSM Size Planner's Fuzzy Logic [PUTN87, DACS87] is a commercial software size estimation package that provides a data base of historical data and uses a sizing by analogy approach. Eleven different application categories are available and each application category is statistically related to a size range. The application categories include business software, avionics, real-time, microcode/firmware, etc.

Three inputs are required by the user:

1. Application category.
2. An overall size category within the application category, chosen from a range of six categories from small to very large.
3. Another selection of size within the size category selected (in 2) above to refine the range of an estimate. Four choices are available for this selection with a range from low-high.

A report using data from the QSM data base is provided. This report is based on the user's selections of application type and size categories. A combined weighted estimate based on conditional probabilities from the user selection and the QSM data base is provided.

Strengths of the model include: modest input, early application and probability distribution to bound the range of an estimate; its weaknesses include subjectivity and lack of precision [PUTN87]. Figure 3.4 illustrates the approach.

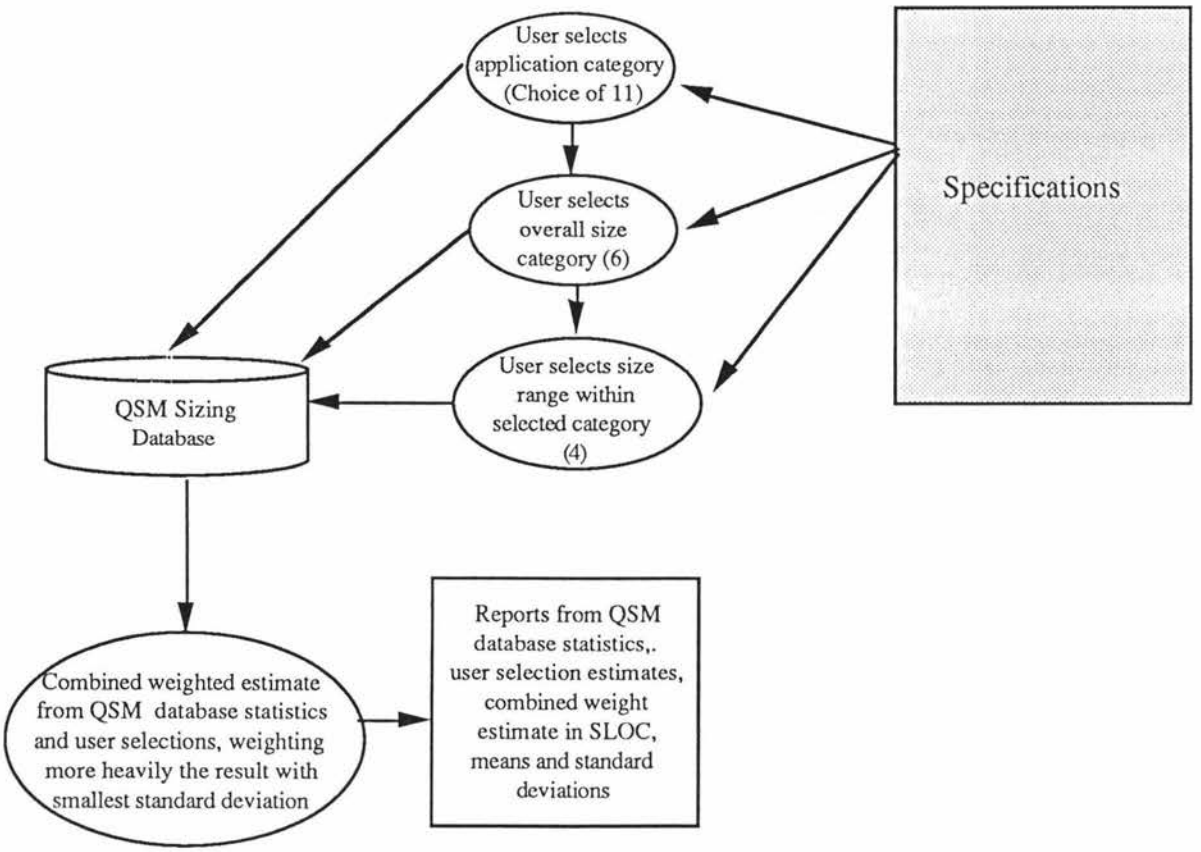


Figure 3.4 QSM Fuzzy Logic

(iii) Other Methods

Price SZ

This package was developed by Price Systems engineering personnel between 1980-1985 [DACS87, OTTE86]. The development was based on the organization's experience with missile and radar projects together with some data from the literature. Because of its ready availability to United States Department of Defense (DoD) personnel, this is currently the most used sizing model by the DoD [DACS87]. The user of the package has a choice of two application types - military or commercial and the size estimate is available in either source or machine level instructions (MLI).

The system estimates size as a function of the following quantitative inputs but details of the algorithms used are not available :

outputs	inputs
pages output to a line printer	message fields
alphanumeric and graphic displays	streams
output streams	operator actions
system states	analogs
computed or created tables	functional bulkiness[1].

Environmental and calibration factors are used to adjust the size to the development environment and organization. These environmental and calibration factors include:

> design review, 'code walk thru', top-down approach, structure/module approach, size calibration factor, language expansion ratio, target size, integration with another system, and program requirements growth.

The user must provide a qualitative rating for many of these factors. The first five of the calibration factors listed above are cost rather than size drivers, with the first four falling into the group of cost drivers Boehm [BOEH81 p.451] terms modern programming practice. Figure 3.5 illustrates the Price SZ approach.

1. Describes software team experience with language and availability of software tools. Normal value = 1, more tools < 1 and fewer tools > 1 [DACS87].

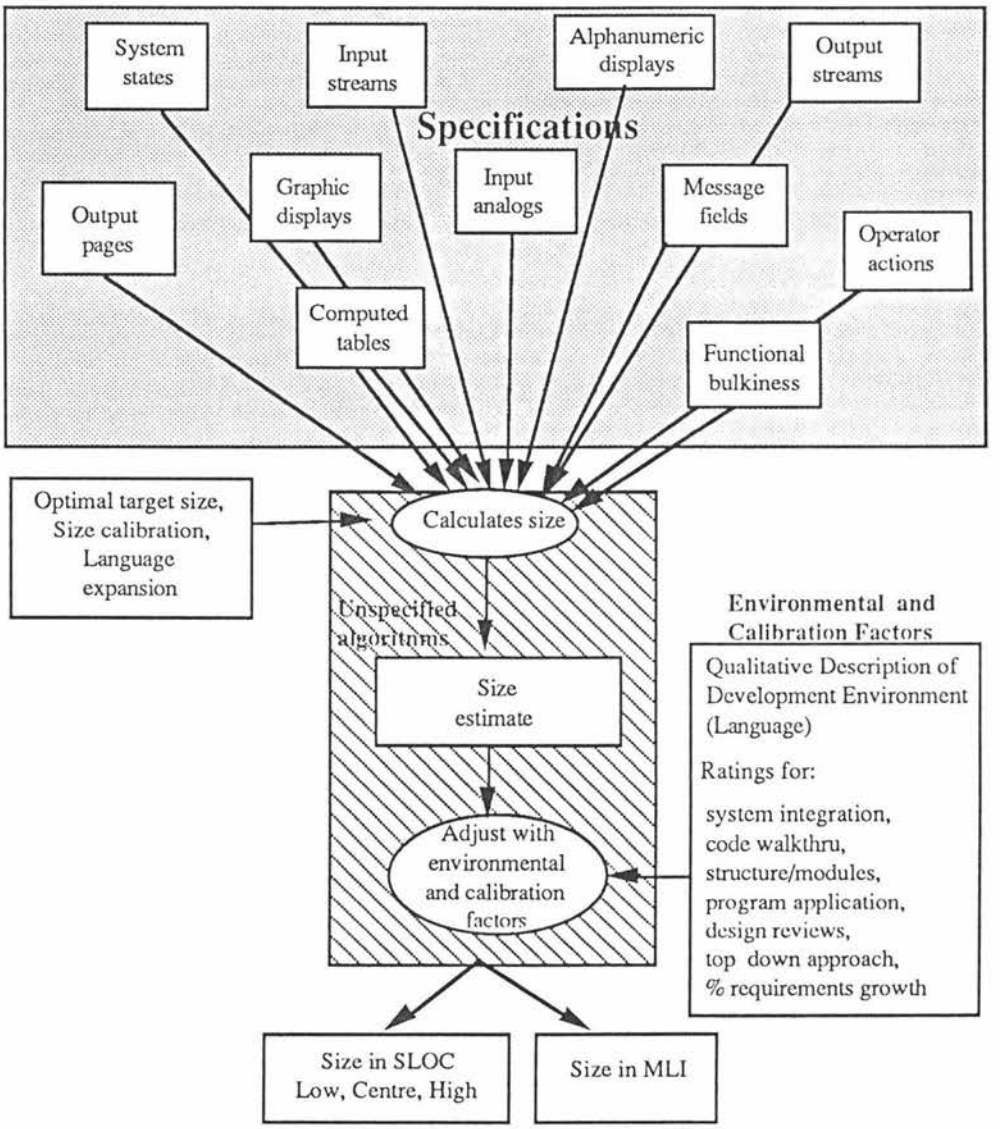


Figure 3.5 Price SZ

3.2.2 System Partitioned into Components

The approaches discussed in this section are those that require the specifications to be divided into components or modules before size can be estimated. Total system size is the sum of component sizes. Boehm [BOEH81 p.318] states that "there is no substitute for a detailed understanding of each software component to ensure accurate software sizing" (thus implying some degree of design - at least enough to divide the project up into component parts). Those estimation approaches that require this view

of the partitioning can be further sub-divided by their component types. One class of estimation methods makes no type distinction between components while the other class partitions the components into several different types.

(A) Single Component Type

This class of estimation approaches requires that the specifications of the system be partitioned into components or modules but makes no distinction between types of component during estimation.

(i) No Individual Component Sizing

The members of this class, with their single component type, do not employ functions based on any objective, measurable properties of individual modules or components in their approach except for perhaps a small number of reference modules, or at most a sample of modules. The following size estimation approaches are grouped within this sub-class:

- Experienced based
  - Delphi techniques
  - PERT based on the beta distribution
  - Other experience-based estimation
- Ranking by size
  - Software Sizing Model (SSM)
  - Curve Fitting
- Analogy
  - Computer Economics Inc. Sizer Planner (CEIS)
  - QSM Standard Components Sizing
- Variable Counts
  - State Machines Model (SMM)

SMM [BRIT85] is an approach similar in some ways to the approach described by Aron [ARON69] which is discussed under the heading 'Other Experienced-based Estimation', however, in Aron's approach, modules are assumed to have an average

implementation size within an organization while SMM obtains its average module size from a variable count for one of the modules. Both SSM [BOZO86] and Curve Fitting [DACS87] have some reliance on expert judgement. In both the user must rank modules (making up the system) by size. Both QSM's Standard Components Sizing [PUTN87] and CEIS [DACS87] rely on analogy with completed projects developed in the same language.

(a) Experience-Based

Although the Delphi techniques described in section 3.2.1 are normally used for system sizing they may be applied at the module level. Putnam and Fitzsimmons [PUTN79 p.191] suggest that a Delphi polling of experts can be used for module size estimates with the following PERT approach which is based on the beta distribution. The early experience-based approach, based on an average module size, described by Aron [ARON69] is also reviewed in this section.

PERT Sizing Based on the Beta Distribution

This is a rather more sophisticated approach than the normal distribution-based PERT approach and involves the separate estimation of each of the software components that will make up the completed software. Three estimates are required for each component to get a range of values that describe the limits, in number of source lines, for the size of each module.

The PERT estimates for each component are as follows:

$$E_i = \frac{a_i + 4m_i + b_i}{6} \qquad \sigma_i = \frac{b_i - a_i}{6}$$

a_i = lowest possible size of the component
m_i = most likely size of the component
b_i = highest possible size of the component.

The estimated total software size then becomes

$$E = \sum_{i=1}^{n} E_i$$

with the standard deviation

$$\sigma_E = \sqrt{\sum_{i=1}^{n} \sigma_i^2}$$

Boehm [BOEH81 p.319] believes that the reason this technique does somewhat better than the PERT sizing of complete systems based on the normal distribution, is because more thought is required when software is broken up into components, and three estimates are made for each component. However, he also suggests, that the standard deviation may be quite misleading as the method assumes that the estimates are unbiased toward either underestimation or overestimation. He is convinced that bias may be present, and suggests that current experience shows that most likely estimates (m_i) tend to cluster toward the lower limit, while actual product sizes tend toward the upper limit. This imparts a significant bias toward underestimation in PERT sizing. Conte [CONT86 p.216] suggests that if the values used for a_i, m_i and b_i are the averages from several different estimators that improved results can be expected. Figure 3.6 illustrates this approach.

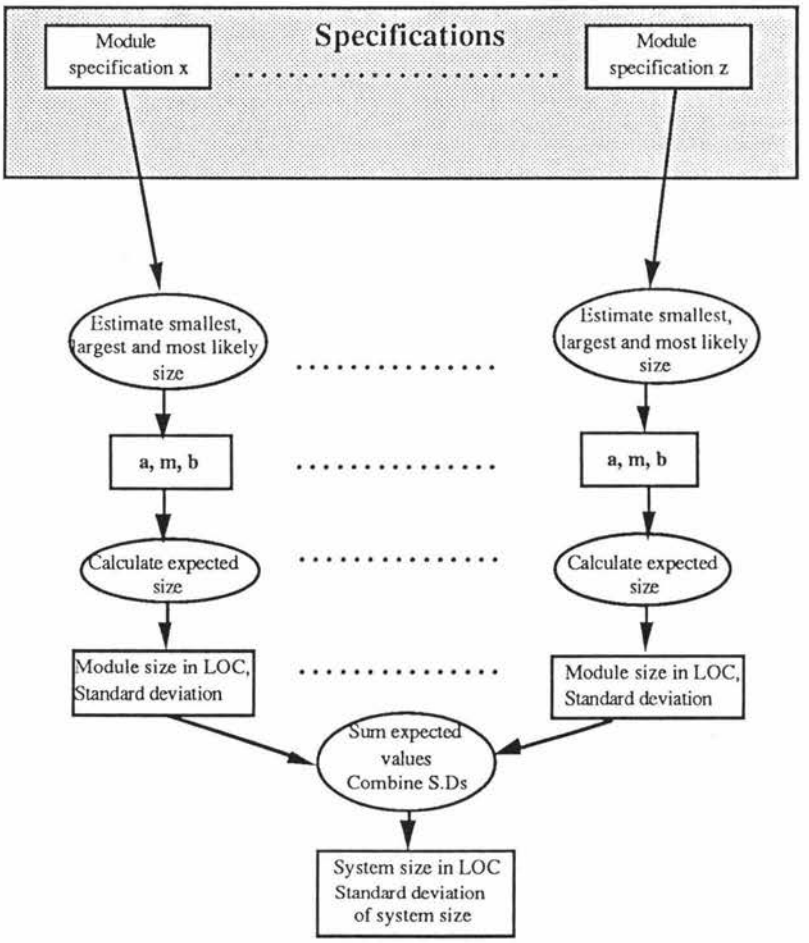


Figure 3.6 Beta Distribution-Based PERT Estimation

Conte [CONT86 p.216] suggests that, although the PERT techniques are an improvement on simple expert judgement, they are still highly subjective and agrees with Boehm [BOEH81 p.320] that experience has shown that most experts err on the low side and quotes an unpublished Yourdon report which showed that several experienced managers underestimated on 12 out of 16 projects when given the complete specifications of the projects on which to base their estimates [CONT86 p.214].

Other Experienced-based Estimation

Aron [ARON69] describes an expert judgement method which is based on partitioning of specifications. The number of units, programs or packages, is estimated from the design by carrying at least one package down to the unit level. The number of units in packages is estimated by experience with similar packages,

sketching the design for the package, or using the same number obtained for the base package. The system size is determined by counting the number of elements and multiplying the result by the average element size i.e.:

Deliverable instructions = number of units x average unit size

Aron suggests that the average unit size depends on the operating habits of the designer and that as an individual tends to normalize his design the resulting units fall within a narrow range of sizes. This enables the designer to determine the average unit size that he assigns to people. This average, he suggests, is perfectly adequate for the purposes of a quantitative estimate.

(b) Ranking by Size

Curve Fitting

This model, developed by Dean in 1983 [DACS87], is purely statistical in nature and relies on the ability and experience of the user "to rank and predict the size of individual functions". The specifications for the proposed system are partitioned into functions. The size of at least three individual functions must be known or must be predictable. All functions are ranked from largest to smallest in any suitable metric. The observed points are input into a curve-fitting model and the unknown points are predicted by fitting the observed points into one of the different curves. The analyst will need to evaluate the results and reject any that are obviously unsatisfactory.

Software Sizing Model

SSM, a computerized simulation model which interfaces to a number of costing models, was developed in the United States by Dr. George Bozoki in 1980 and later refined [BOZO86, DACS87]. This approach, which is based on module comparisons, uses expert judgement together with PERT techniques and relies on the user's ability to rank the size of the modules that make up the specifications. The model may be applied at any stage so long as the software can be partitioned into modules. Bozoki believes that estimators can make more reliable estimates of the relative sizes of modules than they can of their absolute sizes. Two modules must be of known size. These two modules are used as a reference from which to estimate the sizes of the other modules in the set. The two reference modules do not have to

be from the system being developed, any existing modules of known size can be used as a reference.

The modules in the proposed software are ranked using their relative expected size, each is provided with a size range (lowest, most likely and highest), and each is associated with a designated size interval. From a unique pairing of all the modules making up the project the user must judge which is larger of the two, for each pair. The model uses this data to produce module sizes, standard deviations, total system size and probabilities in the unit used for the reference modules (source LOC, function points, etc.) but for source LOC estimates the reference modules must be in the same language as the proposed system. Figure 3.7 illustrates the SSM approach.

If the system being sized is made up of a large number of modules then the procedures can become quite tedious [DACS87] with the following number of inputs required for n modules:

Pairwise comparison data:	(n/2) (n-1)
Ranking data:	n
Sorting data:	n
PERT sizing data:	3n

For 12 modules a total of 126 values per analyst will need to be input into the model.

[DACS87] gives a comparison of results produced when the same data was input into SSM and was used to manually derive PERT size estimates for each of four analysts. The SSM size estimate produced, in this case, was better than that produced by any of the analysts.

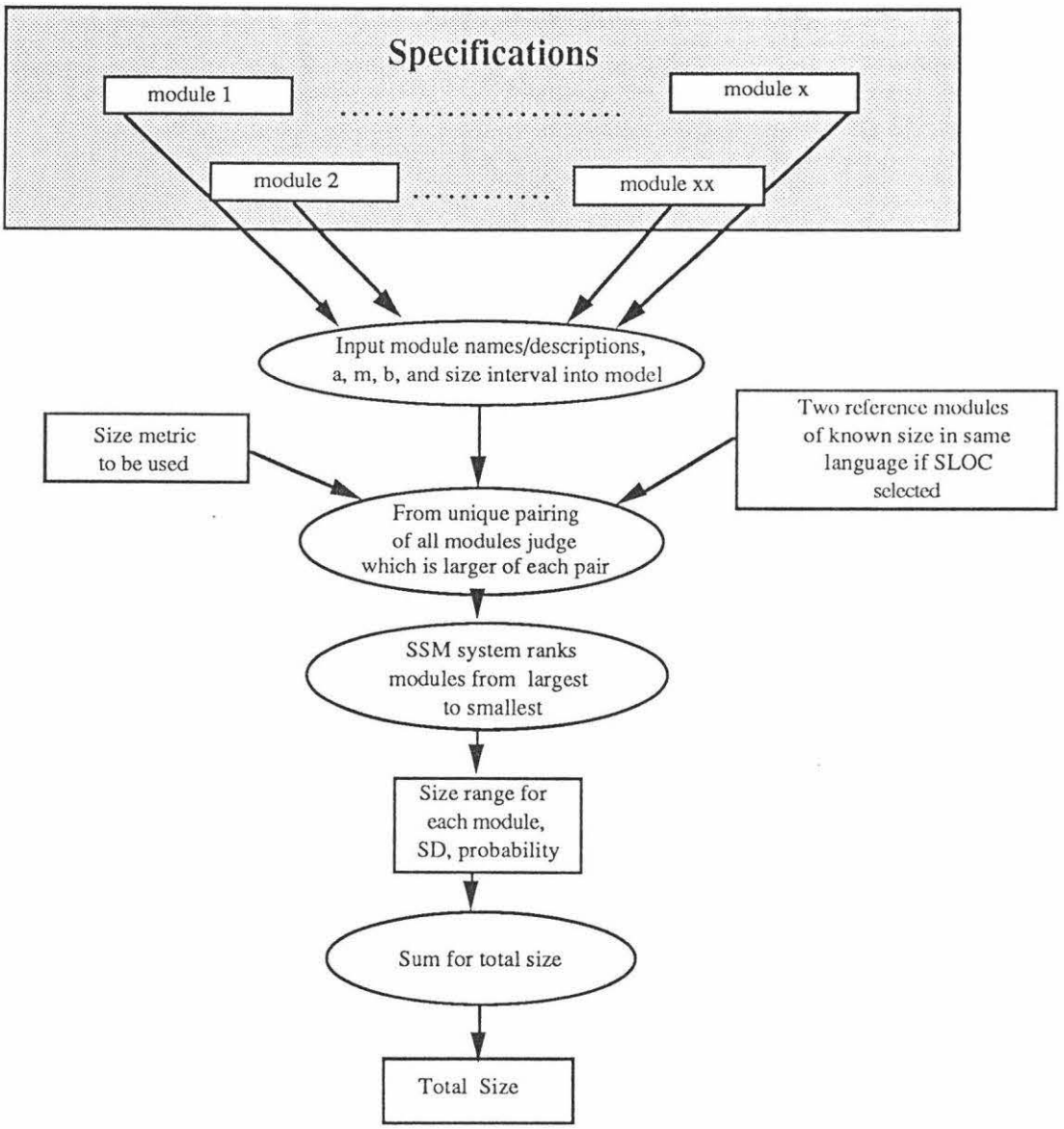


Figure 3.7 Software Sizing Model

(c) Variable Counts

State Machine Model

This approach is a conceptual method classified by [DACS87 p.i] as having a linguistic approach. Britcher and Gaffney [BRIT85] have suggested that during design and implementation any system should be decomposed into six levels of detail

boxes. Level zero is the initial system specification while level five is the source code level. They suggest that on average each box per level would contain about the same amount of function. At the lowest level each function box would be implemented in about the same amount of code. Larger systems would have more boxes at each level not larger boxes. The size of any one function could be estimated by specifying the variable count (amount of data used and generated). The variable count corresponds to Halstead's n_2, operand vocabulary size. To estimate at the procedure level the following formula was used in [BRIT85 p. 108] to estimate the number of assembler LOC:

$$S = \frac{4.8078}{N} V \log_e V$$

where

V = variable count,

N = 1 if V < 100

N = 2 if 100< V < 1000

N = 3 if 1000 < V < 10,000

The result is divided by an expansion value to reflect the use of a higher order language. [BRIT85] used a value of 1.2 to reflect the use of a mixed assembler-macro language.

Once the size is calculated for one procedure-level box then this size can be multiplied by the number of boxes at this level in order to estimate the system. When the approach was applied to a FAA system an average value of six was found for the variable count at the procedure level [BRIT85]. This average value allowed the estimation of average component size at higher specification levels.

An earlier version of this approach was described in 1981 when Gaffney [GAFF81] suggested that the size of modules could be estimated individually using the software science length equation $n_1 \log_2 n_1 + n_2 \log_2 n_2$ with a "standard estimate for n_1, the size of the instruction repertoire", and $n_2 = *n_2$.[1] $*n_2$ would be estimated from the top

[1] n_1 = number of unique operators, n_2 = number of unique operands, $*n_2$ = input/output data item counts

level design of the module or procedure. In practice $*n_2$ might be obtained from an automated design tool. A smaller relative error was found if the formula was applied to the systems they studied in their entirety than if they applied the formula to each procedure individually. Figure 3.8 illustrates the SMM approach.

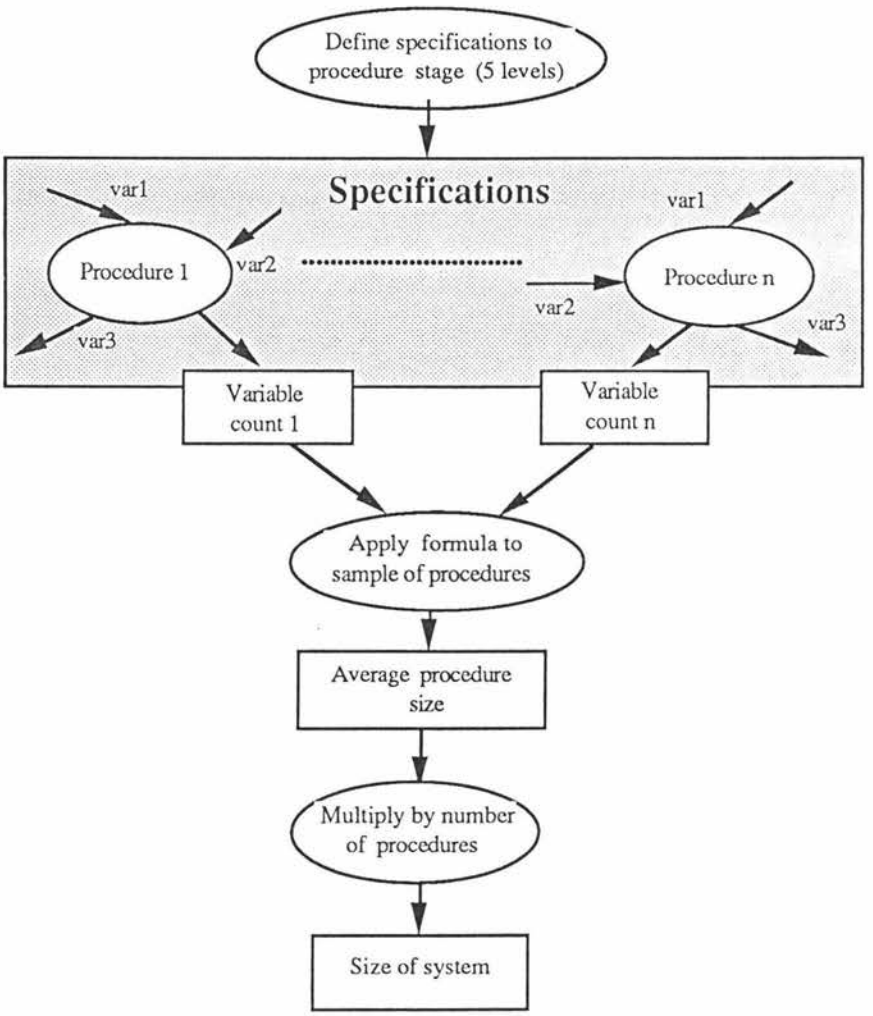


Figure 3.8 State Machines Model

(d) Analogy

QSM Size Planner - Standard Components Sizing

The standard components sizing version of QSM's Size Planner [PUTN87, DACS87] is an approach that estimates the size of a system by analogy with previous projects developed in the same language. Users can use their own historical data base or the QSM data base or have results computed from a weighted combination of the two data bases. The user inputs language and values (low, most likely and highest) for a selection of 'components' from the following list :

SLOC	bits
object instructions	words
bytes	modules
files	subsystems
reports	interactive programs
batch programs	screens.

Figure 3.9 illustrates the QSM SCS approach.

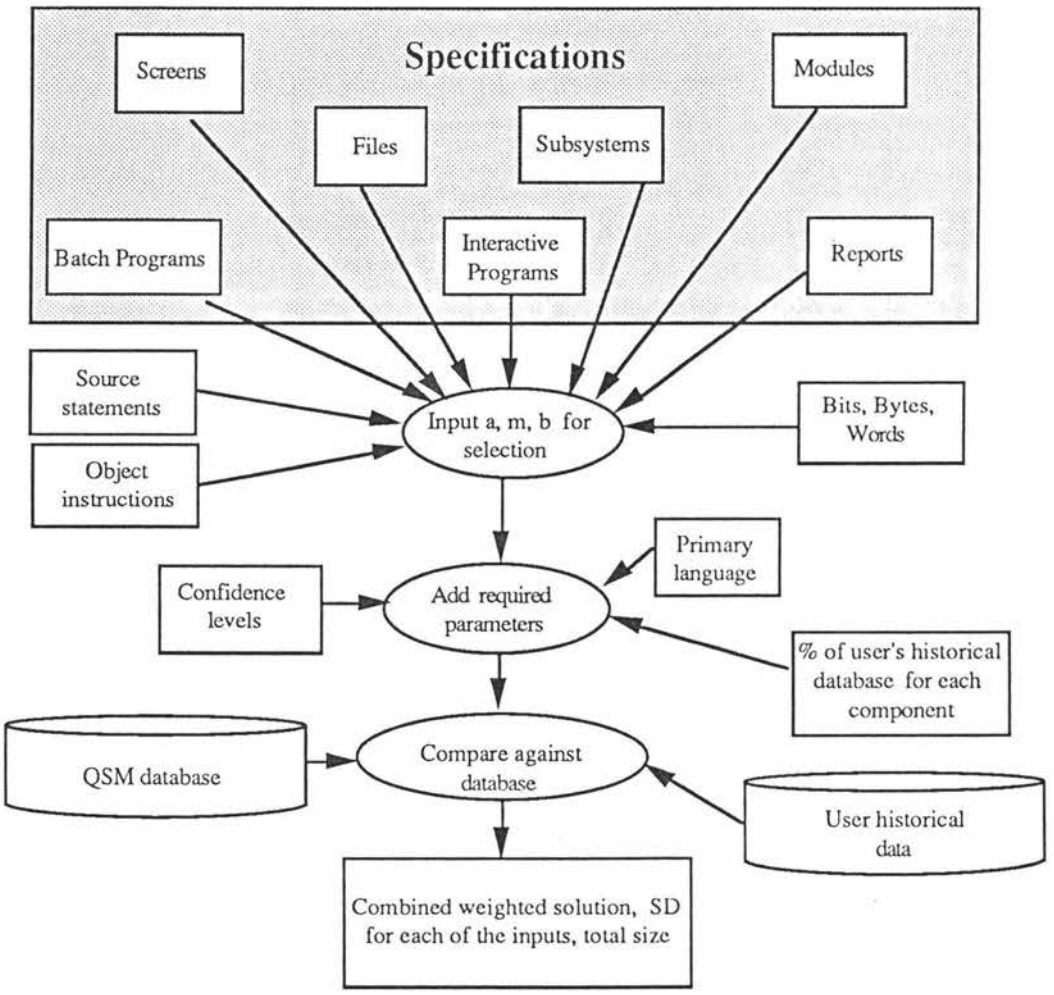


Figure 3.9 QSM Size Planner: Standard Components Sizing

Computer Economics Inc. Size Estimation System

This approach relies on a subjective comparison of the attributes of a new task against the same attributes of three reference tasks of known size in the same language. Figure 3.10 illustrates the CEIS approach.

Three steps are involved in the estimation of a new task's code size [DACS87 p.2-35]:

> Six important project variables are chosen. A pairwise comparison matrix, based on the effect of these variables on project size, is formed. Eigenvalue/eigenvector analysis is performed on this matrix to obtain a relative measure of matrix consistency and weights for each project attribute.

The relationship between the three reference tasks for the same set of project variables is determined by forming a pairwise comparison matrix. The relationship of the new task to each of the three reference tasks for each of the attributes is established.

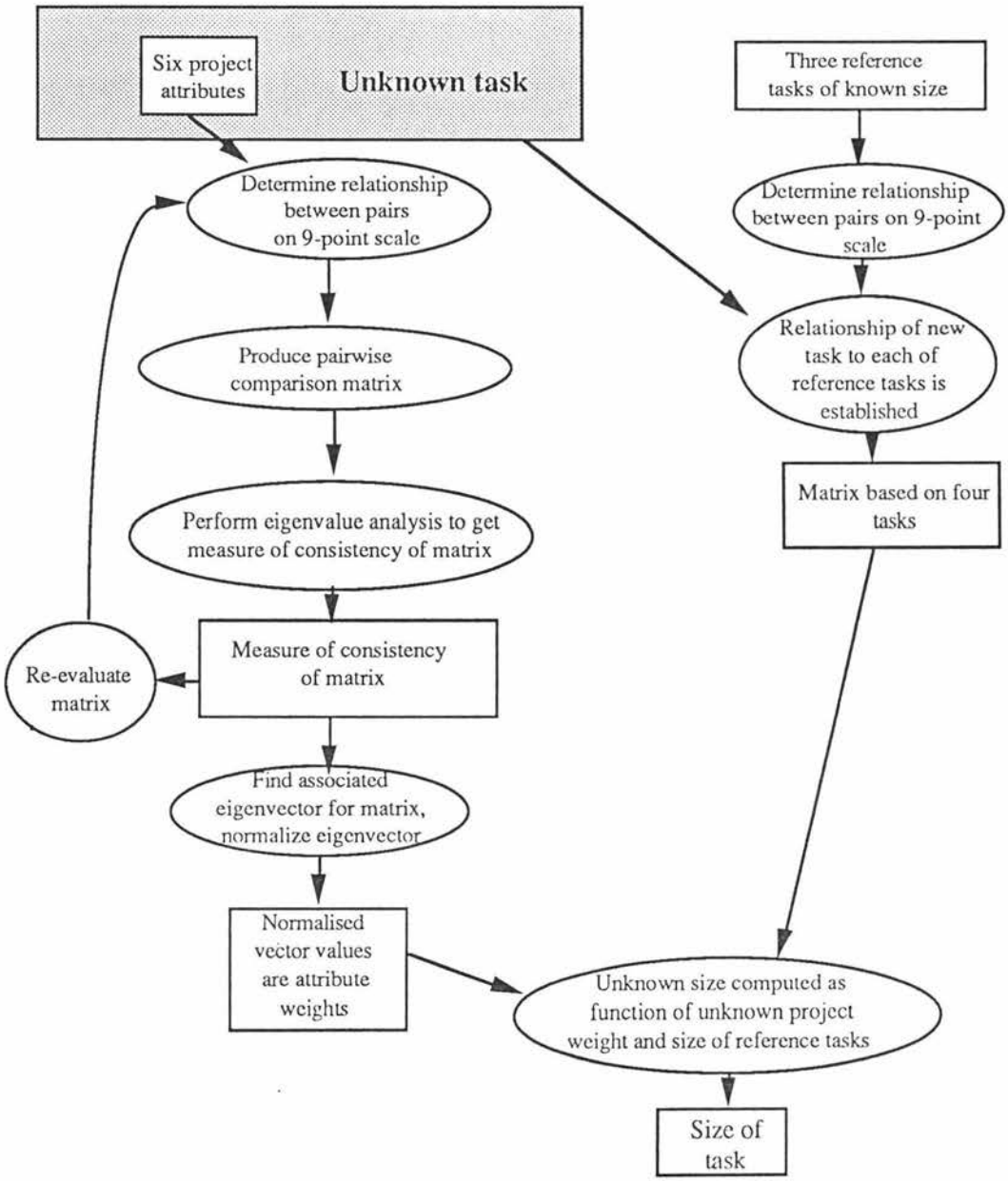


Figure 3.10 CEI Sizer

The unknown project size is computed as a function of the unknown project weight and actual sizes of the three reference tasks.

(ii) Individual Component Sizing

The approaches that fall into this class are those that make an attempt to get the size of each module from some kind of objective measures, usually counts of objects or variables within detailed specifications or design. MarkII [SYMO88], which sizes a single module type from the numbers of input data elements, output data elements and entities is included here together with an approach described by Wang [WANG84, WANG85] that can be used to estimate the size of programs. Wang's approach uses the number of unique variables identified during early program design.

(a) Single Component Type

MarkII

Mark II, which is illustrated in Figure 3.11, is based on Function Point Analysis, and was introduced in section 2.5.2. With its use of a single component type Mark II is here classified differently from the other FPA-like approaches, which use several component types.

Symons [SYMO88 p.2] states that the task of developing a business computerized system is determined by a product of factors, including an information processing size, and a technical complexity factor. The information processing size (raw function points) is defined as "some measure of the information processed". The technical complexity factor is analogous to Albrecht's [ALBR83] adjustment factor, that is "a factor which takes into account the size of the various technical and other factors involved in developing and implementing the information processing requirements" [SYMO88 p.2]. These "factors are intrinsic to the size of the system in the sense that they result directly from the requirements for the system to be delivered to the user". In the Mark II approach all the logical transaction instances making up an application are estimated in the same way, with the size of each transaction being calculated from the numbers of input data elements, output data elements and entities

(replacing the older FPA file concept) used by the transaction. The logical transaction instance sizes are summed to give the information processing size and the technical complexity factor is then applied to give the size of the system.

To obtain the raw function points the counts of input elements I_k, entities E_k and output elements O_k referenced in the kth transaction are summed over all transactions using the formula

$$\sum_k (0.44I_k + 1.67E_k + 0.38O_k)$$

in which the technology-dependent weights shown for the elements relate to the systems studied by Symons. Symons proposed the above weights for the types of systems, normative technologies and environments he investigated but suggested that any other technology would have its own weights and that changes in technology would be shown by changes in the weights.

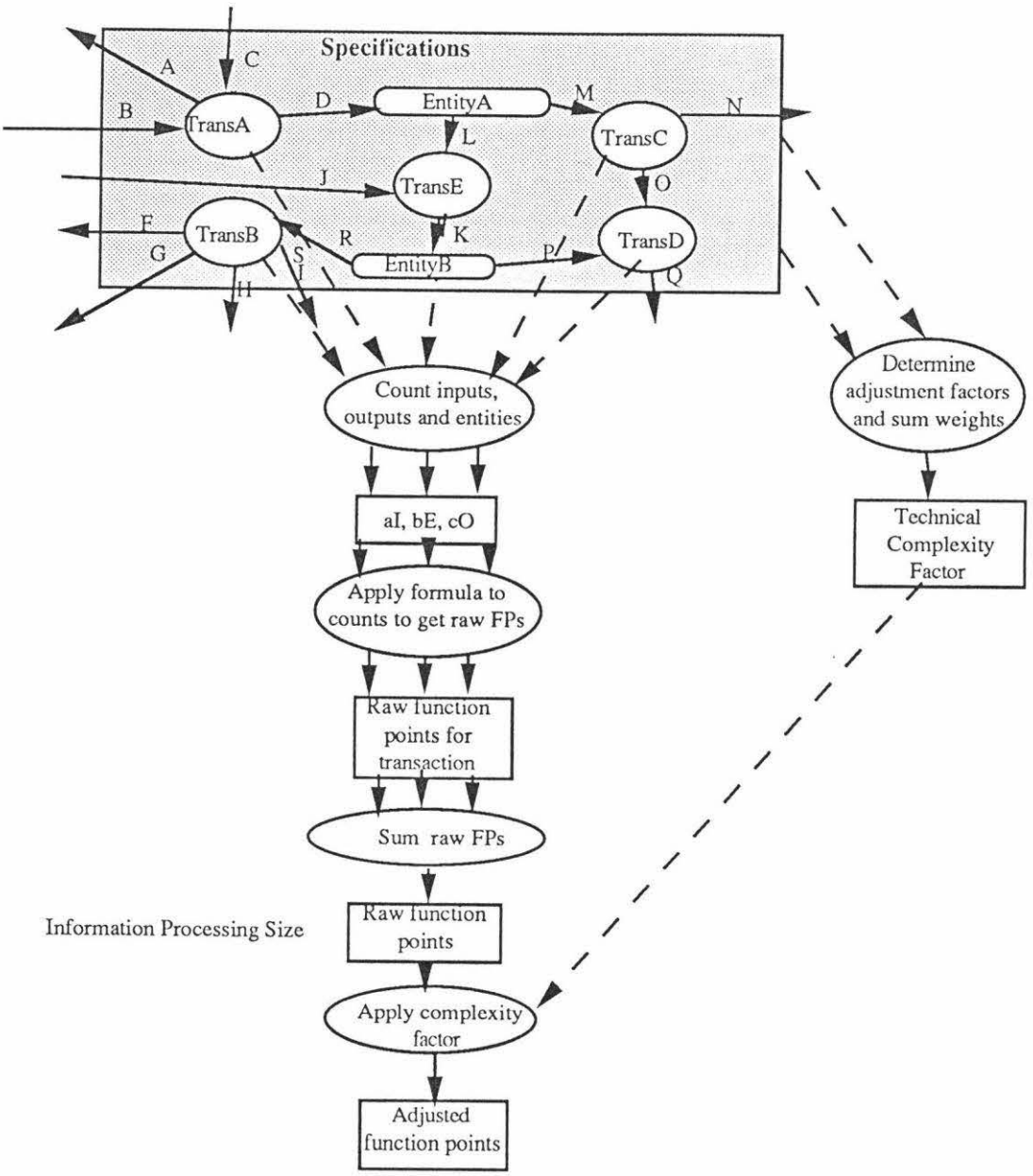


Figure 3.11 MarkII

Symons also suggests [p.8] that the 14 adjustment factors used in FPA are too restricted, that other factors may be needed now, and still other factors in the future, and that some of the FPA factors overlap [p.4]. The restriction of the FPA weights to the 0-5 range may be simple but it is too restrictive and a more open-ended approach to weighting these factors, which would vary with the technology, would appear to be desirable. He used 20 technical complexity correction factors in the examples in

[SYMO88] including one for documentation. MarkII really measures something different from just the software size. It is attempting to estimate the total effort of the job to be done rather than the size of the software. For comparative purposes MarkII has been included in Table 3.1 with the other FPA-based approaches.

Other Approaches

Wang [WANG84, WANG85], in his investigation of Pascal programs, used an approach which he called a data-structure-oriented size estimation method. He stated that a general approach to early size estimation involves two steps: a) establishment of a size model that is a function of some measurable program quantity and b) the identification of development strategies that encourage the early development of such quantities. He applied linear regression to his data set and identified the following relationship:

$$S = 102 + 5.31\ VAR$$

VAR is defined as the number of conceptually unique variables and it can be determined from data-structures and variables developed early during program design. He found that this equation gave better results with his large Pascal programs than Halstead's length equation.

This work is been continued by the Software Engineering Research Center (SERC) with Conte [CONT88] investigating two further size estimation models:

$$S = C * SVAR \quad \text{and} \quad S = C * DVAR$$

where:

SVAR = a count of the unique variables identifiable at the specification stage from a DBMS sub-schema and

DVAR = a count of the unique variables identifiable at the design stage.

C is an expansion factor that depends on language and size level.

Preliminary results when investigating small systems gave values for C with DVAR from 11.8 for Fortran programs to 25.8 for C programs and 16.8 for COBOL programs. A value of 100.6 for C was given for small COBOL programs with SVAR.

(B) Several Component Types

This class requires specifications to be partitioned into different component types. The first sub-class includes those approaches that obtain some kind of average size estimate for each component type; no attempt is made to separately size each individual component within a class. The other sub-class includes those approaches where an attempt is made to size each individual component within the class. This individual component estimation is usually based on a function of counts of some objects within the specifications for each particular instance of each component type.

(i) No Individual Component Sizing

Two FPA-based approaches, SPQR Sizer/FP [SPR86] and Feature Points [DACS87] are included in this sub-class together with ESD [DACS87], an approach which uses analogy with an historical data base of completed components grouped into 105 different function types.

(a) FPA-like

SPQR Sizer/FP and Feature Points

These two approaches were developed at SPR Inc. and are based on Albrecht's 1983 [ALBR83] version of function point analysis which uses several different component types. However all components within a particular component type are given the same (average) weighting. Both Sizer/FP and Feature Points use just three parameters to form the system adjustment factor. These are:

complexity of the problem
complexity of the code
complexity of the data.

These parameters are given subjective ratings on a scale of 1 - 5 by the user.

SPQR **Sizer/FP** uses the five standard FPA component types. SPR have suggested that the effect on the function point total of using the average weights instead of three level complexity weights for each component type is a value that is within 2% of the later (current) Albrecht version of FPA [SPR86] and that as a result Sizer/FP requires rather less effort to produce an estimate. The DACS Report [DACS87 p.4-23] gives an example where Sizer/FP and BYL (Before You Leap, an automated cost estimation tool that includes an FPA front end) [GORD87] were applied to the same system, along with a number of other approaches. Sizer/FP gave a 60% greater number of adjusted function points than BYL which uses a close implementation of the Albrecht 1983 function point method.

The **Feature Points** approach was also developed at SPR Inc. to overcome problems that were perceived when function points were used to estimate real-time, embedded, military and system software. The approach differs from Sizer/FP in two respects:

the weight applied to data files is reduced from 10 to 4
the number of component types is increased to six, with the inclusion of algorithm as a component type.

Table 3.2 includes component weightings for both Sizer/FP and Feature Points. The same three parameters used in Sizer/FP to produce the adjustment factor are used in this method and the adjustment can alter the raw feature point count by 0.6 to 1.4. This approach was under test in 1987 [DACS87].

Both Sizer/FP and Feature Points have been included in Table 3.1 with the other FPA-based approaches. Both approaches, like many of the other FPA-based approaches, are normally used to give size estimates not in function or feature points but LOC. A language expansion ratio is used to convert the number of function or feature points into equivalent LOC for the implementation language. The language expansion ratio provides an average number of LOC required to implement a function point in the chosen language. Table 3.3 summarizes recent research on language expansion ratios.

(b) Analogy

ESD Software Sizing Package

This model is under development at ESD, Hascom Air Force Base, with the developers being primarily Dean and Mentzel [DACS87]. The ESD approach bases the expected size of a module or sub-system on an historical data base of modules in the same language, that perform a similar function. The ESD sizer consists of two primary components, a sizing data base and a user interface that allows a user to retrieve data and produce statistical reports on the selected data set. The data base contains reference data on modules by function type and number of source lines, computer used and language used. The data entries are grouped into approximately 105 functions each of which is identified by an index number. The functions are divided into two groups: operational and support functions. To obtain an estimate for a module the user selects entries from the data base with one or more of the following parameters:

index number	language
system name	function name
range of source LOC	development status
development computer.	

Several parameters may be combined to give more stringent criteria for selecting entries. A temporary data file of the selected entries is created. Statistical analysis is used to obtain the record count, mean, median, variance and standard deviation of the selected data set. The program will also produce a beta distribution curve to yield the most likely value for the new function based on the historical data from the data base. Modules implementing functions that cannot be assigned index numbers must be estimated in some other way. Figure 3.12 illustrates the ESD approach.

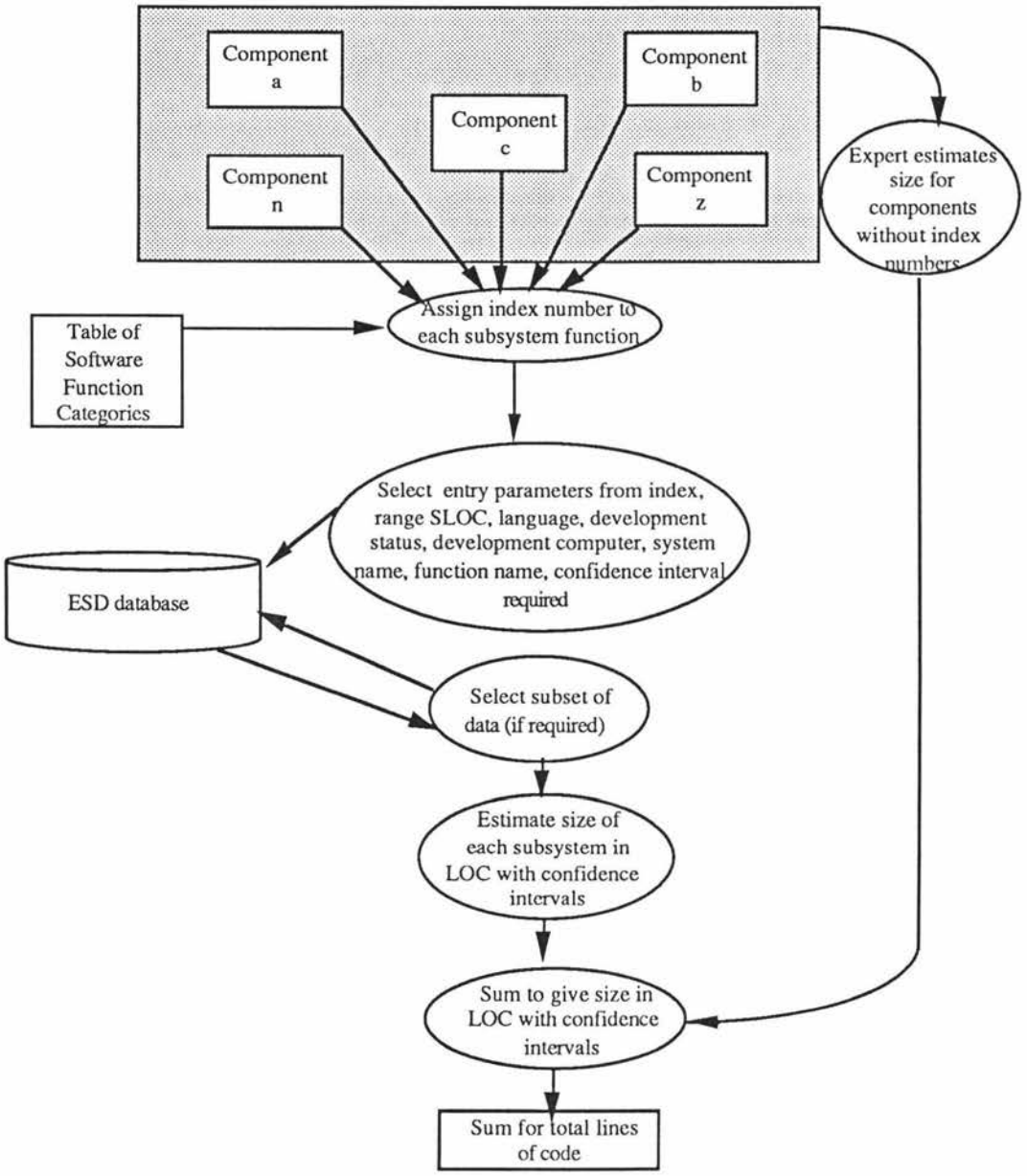


Figure 3.12 ESD Sizing Package

(ii) Individual Component Sizing

The approaches included in this section all use some type of individual component sizing, often from counts of objects occurring within the specifications of the different component type instances although the component sizing in SSA is rather more subjective than that of the other approaches within this class.

This class includes another group of techniques based on Function Point Analysis. Albrecht's FPA itself is included here [ALBR83] and two other FPA-like or FPA-based approaches, ASSET-R [REIF87, DACS87] and QSM FP [PUTN87, DACS87], are also discussed in this section. A somewhat different approach, developed by Itakura and Takayanagi [ITAK82], is also included in this group. This approach was developed in an attempt to estimate the size of reporting programs in a batch environment and although it is very limited in scope, it does attempt to size a program (component) class. BANG [DEMA82, DEMA84], which was discussed earlier, in section 2.5.2, when software size metrics were reviewed, is also included here. SSA, which is also in this group, uses an approach based on an historical data base of previously developed functions. SSA provides 33 function (component) types from which to choose. After data on the function of interest is retrieved, user expert judgement is required to assess the unknown function's size relative to the retrieved functions.

The following approaches are those within this class:

FPA-like	BANG
FPA	SSA
ASSET-R	Itakura and Takayanagi
QSM FP	

(a) Function Point Analysis and FPA-like Approaches

A number of variants of the function point analysis approach to software size estimation have been developed. Approaches like BYL [GORD87], QSM FP [PUTN87] and ASSET-R [REIF87] use FPA or variants to obtain a size in function points. These approaches then use a language expansion ratio (see also 3.2.2Bi (a) and Table 3.3) to convert the function point count to LOC.

Function Point Analysis

FPA was introduced and discussed in some detail in 2.5.1. The function point measure is said to be based on the user's view of the system and it is claimed that

non-DP users can evaluate the measure [RUDO83]. FPA was used by Albrecht to compare productivity between projects that were written in different languages and used different technologies. He used the average number of LOC required to develop a function point to show the relative productivities of COBOL, PL/1 and DMS/VS [ALBR79]. Other writers [RUDO83, JONE86, VERN88, LIM89] have used FPA to produce productivity figures for other languages including 4GLs.

ASSET-R

This approach [REIF87, DACS87], which is an extension of Albrecht's [ALBR79, ALBR83] work, uses three different component and weighting sets depending on the type of system to be sized. Three different system types are available: data processing, scientific, and real time. ASSET-R uses a modified number of components to try to improve prediction accuracy and to extend the approach to real time and scientific systems. Data processing system estimation uses the same five components as the 1983 FPA approach but Reifer infers that the number of weighting factor levels for a component has been extended from the original three to six. There is however, no difference between the weighting factor levels, for very high and extra high, for any of the component types, within any of the system types. This effectively means that there is a maximum of five weighting factor levels in an ASSET-R component type at present. The weightings used by ASSET-R for its three different application types are summarized in table 3.2.

Component operating modes have been added to try to overcome the black box nature of FPA for both the scientific and real time system types. These are defined as "unique modes of operation which are performed internal to the system. Types are dialogue (batch, interactive and interpretive), processing (diagnostic, calibration and execution) and error modes (checkpoint/restart and reconfiguration)" [DACS87 p.A-3]. Another two components, stimulus/response relationships and rendezvous have been included for real time systems.

An extra item, the normalized operator/operand count (OOC_n), is included in the basic ASSET-R calculation. As this item, which is based on Halstead's work [HALS77], may be difficult to calculate before implementation the user may input the number of algorithms instead. Assert-R also includes an architectural constant (ARCH) which deals with the degree of centralization or distribution of the application and its computer(s). This constant ranges from 1 - 2 and has six possible values [DACS

p.A-6]. ASSET-R also includes a reuse factor (rf) to cover a range of low (essentially none) to very high (over 30% reuse) but its derivation and application is internal to the model. The equation used in ASSET-R is as follows:

$$\text{Size (SLOC)} = (\text{ARCH})\ (\text{EXPF})\ ((\text{LANG} * FP_A) + OOC_n)^{rf}$$

ARCH architectural constant	EXPF technology expansion factor
LANG language expansion ratio	FP_A adjusted function point count
OOC_n normalized operator/operand count	rf reuse factor

The type of system being developed will determine the weightings of the parameters in the above equation. The complexity adjustments for raw function points are internal to the model and are different from the normal FPA adjustments. They are not obtained by rating program characteristics [DACS87 p. 2-33].

The technology expansion factor (EXPF) is calculated from nine parameters. These factors are:

requirements volatility	degree of real time code
data base size	environment experience
use of software tools	analyst capability
applications experience	use of modern programming techniques
programming language experience.	

It is worth noting that all the above parameters are included in COCOMO [BOEH81] as cost drivers not size drivers.

FPA-like Software Sizing Approach		FPA	SPQR Sizer-FP	Feature Points	QSM Size Planner	Asset-R	Mark II
System Types	Data Processing	x	x		x	x	x
	Real Time			x		x	
	Scientific			x		x	
Component Types	External Input	x	x	x	x	x	
	External Output	x	x	x	x	x	
	Logical Internal File	x	x	x	x	x	
	External Interface File	x	x	x	x	x	
	External Inquiry	x	x	x	x	x	
	Number of Algorithms*			x		x	
	Operating Modes					x	
	Stimulus/Response					x	
	Rendevous					x	
	Single module type						x
Number of Component Complexity Levels		3	1	1	5	6	NA
Number of Adjustment Factors		14	3	3	0	14#	20

Table 3.1 System Types and Component Types for Function Point Analysis and FPA-like Approaches

Note * in ASSET-R the number of algorithms may be used to get an estimate of complexity, however an alternate and preferred input is normalized operator/operand count.

this is a different set from the usual FPA adjustment factors (see section 2.5.1.)

QSM Size Planner - Function Points

This FPA-like approach [PUTN87] uses the same five component types as the 1983 Albrecht [ALBR83] version of FPA. The major difference between this and the Albrecht version is that there are five complexity ratings (not three) within each component type (see Table 3.2). Another important difference from Albrecht's FPA, is that the raw function point count is not adjusted for complexity by the use of additional adjustment factors. The weighting factors used in this approach to calculate the function point total have not been published [DACS87].

		Very Low	Low	Average	High	Very High	Extra High
FPA							
Data Processing	Input		3	4	6		
	Output		4	5	7		
	Files		7	10	15		
	Interfaces		5	7	10		
	Inquiries		3	4	6		
QSM FP*							
Data Processing	Input	x	x	x	x	x	
	Ouptut	x	x	x	x	x	
	Files	x	x	x	x	x	
	Interfaces	x	x	x	x	x	
	Inquiries	x	x	x	x	x	
Sizer/FP							
Data Processing	Input			4			
	Output			5			
	Files			10			
	Interfaces			7			
	Inquiries			4			
Feature Points	Input			4			
Real-time	Output			5			
	Files			4			
	Interfaces			7			
	Inquiries			4			
	Algorithms			6			
Asset-R							
Data Processing	Input	4	4	4	4	5	5
	Output	4	4	4	4	5	5
	Files	6	6	9	9	9	9
	Interfaces	4	4	6	6	6	6
	Inquiries	4	4	5	5	6	6
Asset-R							
Scientific	Input			4			
	Output	4	4	4	4	5	5
	Files			9			
	Interfaces	3	3	4	4	5	5
	Inquiries	4	4	5	5	6	6
	Operating Modes			3			
Asset-R							
Real-time	Input			1			
	Output			1			
	Files			1			
	Interfaces			1			
	Inquiries			1			
	Operating Modes			1			
	Rendevous			1			
	Stimulus/Response Relationships			1			

Table 3.2 Component Weightings for FPA-like Approaches

* The component weightings are proprietary to QSM

	JONES [JONE86]	SPQR [SPR86]	ASSET-R [DACS87]	BYL [GORD87]	Albrecht [ALBR79]	Other
Basic Assembler	320	320	400	300		
Macro Assembler	213	213				
C	150	128	90	128		
ALGOL	106	105		105		
CHILL	106	105	106			
COBOL	106	105	100	105	110	114†
FORTRAN	106	105	105			
JOVIAL	106		90			
Mixed Languages		105				
Other Languages		105	86			
PASCAL	91	91	70	91		
RPG	80	80		80		
PL/I	80	80	65	80	65	
CMS-2			80			
MODULA-2	71	80		80		
Ada	71	71	72	80		
PROLOG	64	64	64	71		
LISP	64	64		64		
FORTH	64	64		64		
BASIC	64	64		64		
Microfocus COBOL						56*
WANG VS COBOL						60†
LOGO	53	58		64		
English Based Language		53		58		
Data Base Language (FOCUS, RAMIS, IDEAL)	40	40		40		
Decision Support (EXSYS, STRATEGEM, ISPF	33	35		35		
Statistical Languages (SAS, STATPAC)		32				
APL	32	32	38	32		
OBJECTIVE-C	26	27				
DMS/VS					25	
INFORMIX						24*
SMALLTALK	21	21		25		
Menu-Driven Generators (AREV, LINC, PACBASE)		16				14*
Data Base Query Languages QBE, SEQUEL)	16	13				
ALL (4GL)						16§
MULTIPLAN				10		
Symphony/1-2-3				9		
Spread-sheet Languages (LOTUS, MULTIPLAN)	6	6				
Graphic Icon Languages		4				

Table 3.3 Language Expansion Ratios in LOC/FP

† [RUDO83], § [VERN88], * [LIM89]

Other Work Using Function Points

Albrecht and Gaffney [ALBR83] used several formulae, based on software science and function point measures, to get estimates in LOC for some large COBOL and PL/1 programs. This experimental work was conducted to "demonstrate the equivalency of the various measures " (metrics) " and to show their effectiveness as

estimators". They used 24 large programs with four written in PL/1. They found the following independent variables useful when they used linear regression with LOC as the dependent variable:

function points (F), input/output count (V), and function information content ($F\log_2 F$).

Input/output (I/O) count is described as being equivalent to "the total program input/output count without the weights and processing complexity adjustment applied in function points" I/O count and function points are treated "as being equivalent to Halstead's" [HALS77] "n^*_2, the unique input/output data element count" [ALBR83 p.641]. $F\log_2 F$ (the information content) corresponds to $n^*_2 \log_2 n^*_2$ an approximation to the factor $n_2 \log_2 n_2$ in Halstead's length equation.

They then validated the formulae using additional data at three other development sites with another 17 PL/1 programs and found in all cases that the correlation coefficient was >0.92. and concluded that any of the following measures would be a good estimator for SLOC (S):

$S = 73.1F - 4600$ (4 PL/1 programs)
$S = 6.3 (F \log_2 F) + 4500$(4 PL/1 programs)
$S = 66F$ simplified model
$S = 53.2F + 12773$ (based on original 24 cases in COBOL, PL/1 and DMS)

They suggest that software size estimation should be a process which uses function points or I/O count to estimate size in SLOC early in the development cycle and that this method provides a bridge between function points, software science and SLOC. They state that the early availability of elements that comprise function points and I/O count for an application would mean that estimation could be done earlier in the development cycle than the usual SLOC-based estimation techniques.

(b) Analogy

Software Sizing Analyser

SSA [DACS87] is an automated package that uses an approach based on analogy with an historical data base of completed projects, grouped by function, with some

requirement for expert judgement. The data base consists of aerospace data for space systems applications. The data is grouped into approximately 33 standard functions which are identified by a keyword. The data stored for each function type includes: a classification on a five level (1 - 5) complexity scale, language, size in LOC and MLI (machine level instructions). The complexity entry is based on the judgement of the data collector with most entries being of a medium complexity (3).

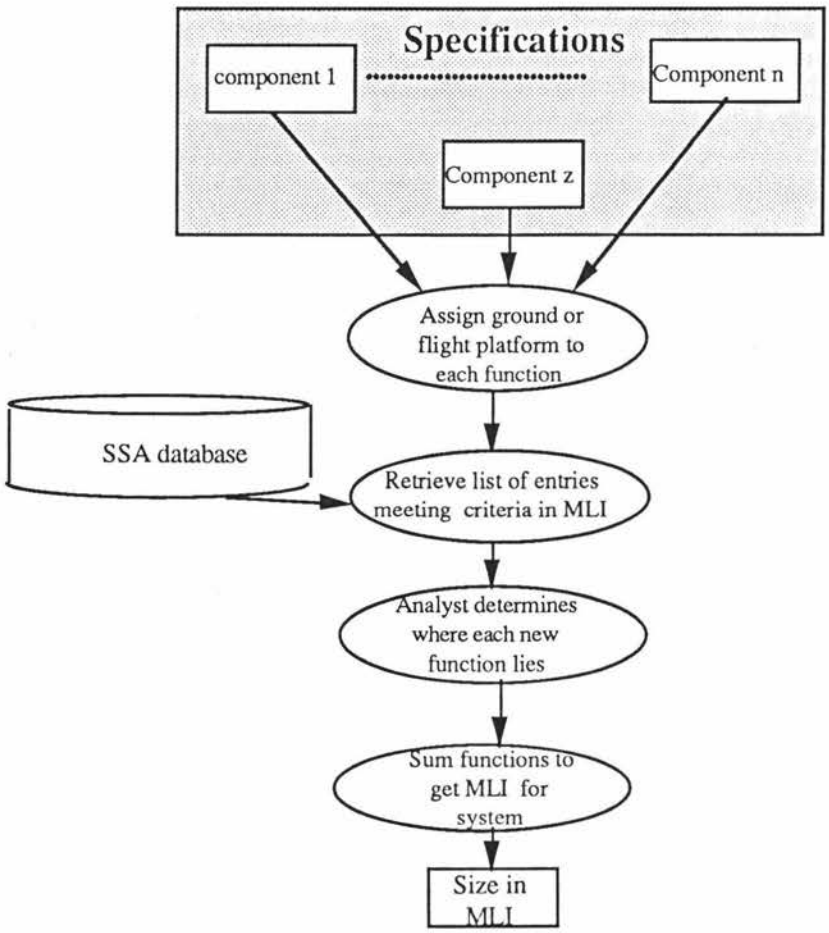


Figure 3.13 Software Sizing Analyser

The approach gives estimated size for a single module performing a named function. To use the model the user must specify whether the unknown function is a ground or flight function together with a function keyword. The model will retrieve a list of entries satisfying the search criteria, with the number of cases being summarized by complexity. The analyst must determine where in the high/low range of instructions

the new function lies thus incorporating some degree of expert judgement. This is an attempt to size individual components with a little more than just an average component size. The SSA approach is illustrated in Figure 3.13.

(c) Other Approaches with Individual Component Sizing

System BANG

BANG was introduced and discussed in Section 2.5.3. The approach is summarized in figure 3.14.

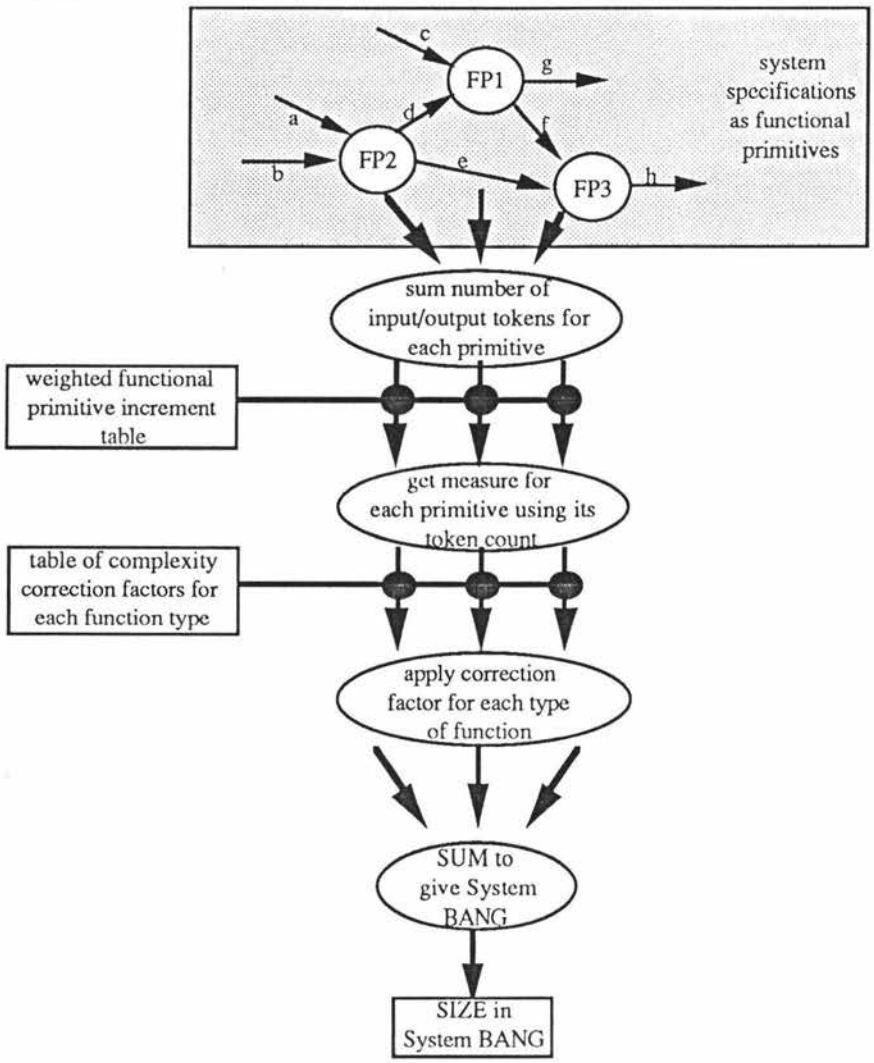


Figure 3.14 System BANG (adapted from [VERN87])

Itakura and Takayanagi

Because of the problems they saw in estimating size as input to algorithmic costing models, Itakura and Takayanagi [ITAK82] developed a method for partial size estimation for a specific kind of program, namely batch reporting, within a specialized type of system. They described it as "a program-size estimation model for batch programs in a banking system". Their model for COBOL programs was developed from input and output items and elements, as well as program processes.

Their equation, for predicting program size, used the following variables:

(1) Number of input files (X_1)
(2) Number of input items (X_2)
(3) Number of output files (X_3)
(4) Number of output items (X_4)
(5) Number of reports (X_5)
(6) Number of horizontal items in reports (X_6)
(7) Number of vertical items in reports (X_7)
(8) Number of calculating processes (X_8)
(9) Existence of sorting (X_9)
(10) Report type (X_{10})

$$\text{where } X_{10} = \sum_{i=1}^{X_5} \begin{cases} 0 & \text{:transaction type} \\ X_{6i}+5 & \text{:transaction, branch total and bank total type} \\ 2X_{6i}+30 & \text{:branch total and bank total type} \\ 2X_{6i}+5 & \text{:bank total type} \end{cases}$$

The variables, which the experts believed affected program size, were selected from information that was available at the time of estimation. Values for the coefficients of the variables in the equation, were determined by experts from the number of lines that each of the variables was expected to require in a program and was based on code in all the COBOL sections (i.e. Environment division, Data division Working Storage section etc.) of a program. This gave a linear equation that was used for estimating the total number of LOC for a program.

$$Y = 105 + 8X_1 + 1.3X_2 + 8X_3 + 3.3X_4 + 131X_5 + \sum_{i=1}^{X_5} (4.6X_{6i} + X_{7i} + 80X_{8i} + 61X_{9i}) + X_{10}$$

where Y = the number of source LOC

Initial estimates were made early, when reports and process flows were tentative, before file formats existed and before "any fundamental design work" had been done. These estimates proved to underestimate by 13% but Itakura and Takayanagi felt that this was quite good considering the poor initial conditions for their estimates, with potentially inaccurate data. The later (better) estimates, which were done at the end of the design stage, showed 7% underestimation which they considered very effective. Thirty eight small to medium sized COBOL programs were used to test their estimation equation. Their model underestimated for programs of over 2500 LOC. Even with the later estimates, the standard deviation was 690 LOC (i.e. 35% when the average LOC was 2000). They did not think that the model could be used directly by others but suggested that their modelling process was generally adaptable to other projects with the selection of other appropriate variables and coefficient values.

The method was intended to give a more objective measure of program size but because of the subjectivity of the coefficients used, it must still be considered a method with a large expert judgement component. Because there has been no other experimental work published about this model, it has not been possible to discover how widely used or how successful it is. Boehm [BOEH84] suggested that this model would "appear to give reasonably good results for small-to-medium sized business programs within a single data processing organization". The method may be useful in an environment where there are well-defined programming standards and where the range of processes is limited. Boehm [BOEH84] noted that the method "is a useful step in the direction" of finding a formula for software size from quantities known early in the software lifecycle.

3.2.3 System Adjustment Factors and Relationship to Component Size Estimation

A system adjustment factor is "a technical complexity factor which takes into account the size of the various technical and other factors involved in developing and implementing the information processing requirements" of an information system [SYMO88 p.1]. It is determined by estimating the 'degree of influence' of one or more general application characteristics to get an overall system adjustment factor which is used to modify the information processing size.

(i) Use of Adjustment Factors in the Sizing Models Surveyed

As shown in Figure 3.2, most size estimation approaches fall within classes where estimation is based on partitioning a system into components. These estimation approaches may take one of two views of total system size estimation. The first group (11 out of 15 of the methods in Figure 3.20) view system size as the sum of component size estimates. Approaches like SSM, SSA, ESD, Curve Fitting, BANG, and CEIS are members of this group and are without subsequent size adjustment. The second, and smaller, group of approaches are almost all based on FPA. These estimate a "raw" system size by summing component size estimates and then adjust this size with an overall system adjustment factor. Some of the software size estimation models that were discussed earlier in this chapter, incorporate several adjustment factor elements that are cost drivers, rather than size drivers, and hence are not appropriate in this context.

Adjustment factors occur in a minority of sizing models. An uncompromising view might regard them as at worst an admission of failure on the part of a model to include their effects in the component sizing process or, at best, the result of intentionally including only particular component types in the unadjusted size measure as, for example, in Albrecht's FPA. However, in the increasingly complex world of software size metrics, they are often a useful way of handling hard-to-measure size drivers not adequately covered elsewhere.

	FPA	Asset R	Feature Points	Sizer FP	Mark II	Price SZ
Number of factors	14	11	3	3	?20	8
Data communications	6					
Distributed functions	6					
Performance	6	5			x	
Heavily used configuration	6				x	
Transaction rate	6				x	
On-line data entry	6				x	
End user efficiency	6				x	
On-line update	6				x	
Complex processing	6					
Reusability	6					
Installation ease	6					
Operational ease	6					
Multiple sites	6					
Facilitate change	6					
Functional bulkiness*						< 1 <
Size calibration factor						.7-1.5
Logical complexity			x	x		
Code structure*			x	x		
Data complexity			x	x		
Distributed archictecture		7				
Requirments volatility/growth*		5				0-18%
Database size		4				
Use of software tools*		6				
Applications experience*		5				
Programming language experience*		5				
Environment experience*		5				
Analyst capability*		5				
Modern programming techniques*		5				
Degree of optimization		5				
Integration with another system					x	y/n
Design review*						y/n
Code walk thru*						y/n
Top-down approach*						y/n
Structure/module approach*						y/n
Security					x	
Direct access for third parties					x	
Documentation requirements*					x	
Special user training facilities					x	
special/unique h/w or s/w*					x	

Table 3.4 System Adjustment Factor Elements and their Ranges or Number of Complexity Levels

* denotes adjustment factor elements identified as cost drivers.

x denotes unpublished value.

The estimation approaches that use an overall system adjustment factor, the adjustment factor elements and number of complexity levels or range for each element are shown in table 3.4 above. The number and weights of the adjustment factor

elements, and the subjectivity of their application in FPA-like approaches were discussed in section 2.5.1 (iv).

MarkII has tried to overcome some of these problems with adjustment factors in FPA-based approaches. The author is only aware of one estimation method, not based on FPA and component partitioning, that uses an overall system adjustment; this is Price SZ which estimates the system as a whole, with an unpublished algorithm, and then applies an adjustment factor to this size estimate. Both Price SZ and ASSET-R can be criticized because they include system adjustment factor elements that are cost rather than size drivers.

(ii) Adjustment Dependence on what Size is Required for what Purpose

In reviewing the need for an overall system adjustment factor it is necessary to determine what kind of size estimate is required and for what purpose. The whole system may be the subject of estimation or there may be cases when a partial size estimate (for example of components in a *functional* specification) is enough with no requirement for anything more. Which adjustment factor elements are appropriate depends on whether a complete or partial size is being estimated and, in the latter case, what partial size, and for what purpose. In some cases where a partial size estimate is required some system adjustment factor elements may not be necessary because certain system features, for example data communications, may either be supplied with the technology or be developed separately by systems programmers. In such a case these system adjustment factor elements may be the subject of a separate project, implemented by a separate group, with minimal impact on the application development project for which the primary size estimate is required.

(iii) Conflicts Between FPA Adjustments and Component Sizing Concepts

The FPA approach bases its component size estimates on some kind of average which is then adjusted up or down by the overall adjustment factor. There is a potential conflict between adjustment factors of this type and the concept of component size

estimation. A size estimation approach that uses a multiplicative adjustment across the whole system, like FPA, assumes that the housekeeping functions will be some proportion of the system size and, all other things being equal, will be larger for systems with a larger component size estimate. This may be true in many cases but need not always be so. In any event, it is suggested that functional module size estimation should be based solely on unadjusted FPs, if it is to be meaningful for individual modules. The idea of adjusting an average value up or down is inconsistent with the idea of module size estimation because it assumes that the adjustment affects all modules equally, or it implies that a downward adjustment can make systems smaller than the sum of their unadjusted module sizes, which seems unnatural. Adjustment factors are discussed further in section 4.5 and an adjustment factor model is presented in 4.5.2.

3.3 A FIRST ALTERNATIVE CLASSIFICATION OF SIZE ESTIMATION APPROACHES

The DACS Report [DACS87] divided software size estimation approaches into six categories "that illustrate general approaches to sizing" [p.2-1]. The classes identified in [DACS87], and the specific methods within these classes, were as follows:

Sizing by Analogy
- ESD Sizing Package
- SSA
- QSM Size Planner - Fuzzy Logic

Size-in-Size-Out
- Wideband Delphi Technique
- SSM
- Curve Fitting
- PERT

Function Point Analysis
- SPQR SIZER/FP
- Feature Points
- ASSET-R
- BYL
- QSM Size Planner - Function Points

Comparison of Project Attributes
- CEIS
- QSM Size Planner - Standard Components Sizing

Linguistic Approach	Other Approaches
ASSET-R	PRICE SZ.
SMM	

The characteristics of these classes are discussed below but other approaches, not discussed in [DACS87], have been included within these classes, where appropriate. Because QSM Size Planner [PUTN87] has three approaches to size estimation it has been included in three different classes. ASSET-R, which is a function point analysis variant, but uses some linguistic ideas, was included by [DACS87] in the two classes shown above.

3.3.1 Sizing by Analogy

This approach relates the proposed development to previously developed modules and systems of similar function and environmental requirements. The analogy approach can be applied both at system and module level. ESD Software Sizing Package [DACS87], and SSA [DACS87] are examples of the module level analogy approach while QSM - Fuzzy Logic [PUTN87], which was classified in the structural classification under 'no partitioning of specifications', is a system level example. System level estimates are based on a comparison with similar application systems.

The use of analogy at the function level, though more time consuming, provides more refined estimates and requires a more detailed assessment of the system components. At the module level the approach involves the use of a data base of previously developed modules. Information stored includes both function within the previously developed software, and size in source lines and/or machine level instructions required to develop each function. Data about various previous implementations of similar functions can provide a range of sizes for the user who selects a subset of items on which to base an estimate for the new function. ESD and SSA both estimate size by analogy at the function level. The major difference between them is the presence of a five level complexity scale that is applied to components in SSA. This attempt to obtain sizes for individual components placed SSA, in the earlier structural classification, into the 'several component types - individual component sizing' class. ESD was placed in 'several component types - no individual component sizing'.

3.3.2 Size-in-Size-Out

Approaches in this class are described as purely statistical and require approximate size estimates, based on expert judgement, as input. They aim to get the most out of the data the experts can supply. The approximate size values used include: size ranges, relative ranking data, estimates of some of the modules making up a system and several independent estimates from individual experts. The Wideband Delphi Technique, PERT, SSM and Curve Fitting are included in this class. Both the Wideband Delphi Technique and PERT may be applied at either the system or function level while SSM and Curve Fitting are both normally applied at the function or module level. In the structural classification PERT, SSM, and Curve Fitting were included under 'system partitioned into components - no individual component sizing' while the Wideband Delphi Technique was included under 'no partitioning of specifications - experienced based'.

3.3.3 Function Point Analysis

FPA was described in detail in sections 2.5.1 and 3.2.2 (iib). The DACS Report includes the following approaches in this category: SPQR SIZER-FP, Feature Points, ASSET-R, BYL, and QSM Function Points. BYL is a close implementation of Albrecht's 1983 version of FPA. Because SPQR SIZER/FP and Feature Points use an average component size for component type instances they were included, in the earlier structural classification, under 'systems partitioned into components - several components types - no individual component sizing'. FPA itself, ASSET-R and QSM Function Points provide different complexity ratings and weights for each component type based on a more objective measure of the component sizes and were hence classified within the earlier structural classification as 'systems partitioned into components - several components types - individual component sizing'. Work on MarkII had not been published until after the DACS Report. MarkII is included in this DACS class because of its origins in FPA. In the earlier structural classification MarkII was the sole member of the class 'single component type - individual component sizing'.

3.3.4 Linguistic Approach

This class applies to those approaches that count the lexical symbols used in the programmatic expression of an algorithm. They can be found in pseudocode expressions of the formulae to be used, or in after-the-event source code of a system. These approaches to estimation are based on Software Science's [HALS77] ideas of operators and operands. ASSET-R, bases its size estimate on a function of the adjusted function points and operator/operand count, while SMM uses a variable count that corresponds to Halstead's operand count n_2. The DACS Report did not mention System BANG but it can also be included in this class as it uses a factor based on an operator/operand count to adjust the size of a functional primitive. Wang's [WANG84, WANG85] data-structure-oriented approach, with similarities to SMM, estimates size from a count of variables based on Halstead's n_2 and is included within this class. Because SMM suggests the use of variable counts to obtain the size of a few modules and then uses an average size for the rest, SMM and data-structure-oriented sizing (in the earlier structural classification), were included within different classes. SMM was placed under 'single component type - no individual component sizing' and the data-structure-oriented approach under 'single component type - individual component sizing'.

3.3.5 Comparison of Project Attributes

This class of methods, like the sizing by analogy class, requires a data base of completed projects. The approaches within this class relate the current project to previous developments at either the system or component level. DACS differentiates these approaches from those in the analogy class by suggesting that this group takes the analogy approach to a more detailed level of specification.

Parameters are identified that correlate the size of the new program with specific attributes and these attributes are compared with the same attributes of completed projects. Statistical analysis is then used to yield an estimated size. CEIS with its matrices and eigenvalue/eigenvector analysis and QSM Standard Components Sizing are classified in this group.

The Itakura and Takayanagi [ITAK82] work was not mentioned in the DACS Report because it gives only a partial size estimate and is not automated. Their approach does not obtain its estimated size from a data base of completed projects. The approach however shows many of the characteristics of this class. However variables in the prediction equation have their coefficients derived by observation from similar completed developments. The Itakura and Takayanagi approach has therefore been included in this class.

3.3.6 Other Approaches

DACS could not explicitly associate one approach, PRICE SZ, with any of the above classes because little is known about the underlying techniques used. It does not employ FPA but some of its inputs are similar to function point parameters.

3.4 A SECOND ALTERNATIVE CLASSIFICATION OF SIZE ESTIMATION APPROACHES

The second alternative classification of size estimation approaches is based on a consideration of how much of the system definition, design and development process must be completed before the earliest application of the method is possible. This classification could also be said to be based on the set of minimal information required before the method can be used and this is, of course, dependent on development phase. Different analysis and design methodologies may be used with different developments or tools. What is available early with one methodology may not be available quite so early with another. It is not the purpose of this dissertation to survey such a large topic as this. Rather, a typical, widely-used methodology is considered. Thus the classification shown below assumes the use of a methodology similar to structured systems analysis [GANE79]. The discussion that follows also makes the assumption that structures identified in the early phases of specification or design will be carried through to later phases of development and will be mirrored in the implementation.

Most of the size estimation approaches can, and should, be applied several times during development with an expected increase in accuracy in the later stages. One

approach that can be used very early, QSM Fuzzy Logic [PUTN87], does not actually need a defined project. Fuzzy Logic can be used to get an early general idea of the size of any system that might be of interest within several classification types.

Using this second alternative classification, the following four size estimation classes are identified:

feasibility
requirements
preliminary design
detailed design.

These classes are based on phase descriptions by Boehm [BOEH81 and Fairley [FAIR85]. Table 3.5 shows these classes (phases), the reviews that occur at their completion and the major documents, of interest here, produced at the end of each phase.

Of course, there are several different development approaches with this table showing just one, the lifecycle approach. Incremental development, spiral development and prototyping will all have different characteristics, phases and reviews. However, no matter which approach to development is used there is a minimal essential information set that is required before a specific software sizing method can be used.

Phase	Phase Review Documents Produced	Components Defined
Feasibility Define preferred concept, determine life cycle feasibility	Product Feasibility Review (PFR) System definition and project plan	problem definition, goals, constraints, justifications functions to be provided, user characteristics, development/operating/maintenance environments lifecycle model preliminary development schedule and cost schedules tools, techniques, programming languages, testing requirements.
Requirements Complete, validated set of required functions, interface and performance.	Software Requirements Review (SRR) Software requirements specification Preliminary user's manual Preliminary verification plan	external interfaces and data flow user displays/report formats user command summary high level data flow diagrams logical data sources and sinks logical data stores logical data dictionary exception handling.
Preliminary Design Complete validated specification, includes data structures and control structures.	Preliminary Design Review (PDR) Architectural design document	data flow diagrams conceptual layout of data structures attributes of data objects name and functional description of each module interface specifications for each module interconnection structure of modules interconnections among modules and data structures exception conditions.
Detailed Design Verified detailed design and specification for each unit	Critical Design Review (CDR) Detailed design specification User's manual Software Verification Plan	physical layout of data structures and data bases, data dictionary specification of all concrete data objects, detailed algorithms.

Table 3.5 Early Phases and Reviews in Lifecycle Model

Figure 3.15 shows a schematic view of the relationship of size estimation approaches to the different phases of the lifecycle. The approaches that may only be used in a phase under certain criteria are marked with a '?'. It must be emphasized again that this summary applies only to this particular lifecycle model and that different placings may result if a different model is used.

3.4.1 Feasibility

Feasibility is completed with the product feasibility review (PFR). The system, goals, functions to be provided and user characteristics have been defined by the time of the PFR when the following software size estimation approaches can be used:

Almost all approaches where the system is sized as a whole except for Price SZ. This latter approach is excluded because it requires details that will not be available until the software requirements review (SRR) (for example computed tables). Approaches included at this stage are:

1) Experience based
   PERT normal
   Delphi techniques
   QSM-Fuzzy Logic

2) Approaches that use analogy with previously developed functions. Although no design has been done, the major functions have been identified. If a data base of completed and categorized functions is available then the size of each of the defined functions can be estimated. ESD and SSA are approaches that may be used in this way.

3) Curve Fitting could be used at this time if there are at least three other suitable completed projects available and it is possible to rank the proposed system against these projects. Alternatively, if some suitable functions of known size are available, then these could be used for estimation using Curve fitting, 'unknown' functions could each be separately estimated, then summed to produce a final estimate.

4) CEIS can be used if the term 'task' describes the project as a whole and three reference projects of known size are available.

SSM was included in the DACS Report [DACS87] with models that can be applied at this stage of development. SSM has not been included here because it uses module

comparisons and this implies design to a stage that will at least identify the required modules.

3.4.2 Requirements

The requirements phase is completed with a software requirements review (SRR). External interfaces, report formats, data flows, user display formats, and logical data stores have been identified by the end of this phase. The FPA-based approaches (except ASSET-R which is excluded because of its requirement for more detailed information about algorithms) and other approaches that use system (not module) inputs and outputs can be used by the SRR.

FPA, SPQR Sizer/FP, Feature Points, MARKII, CEIS and the Itakura and Takayanagi approach can all be used once the SRR has been completed.

3.4.3 Preliminary Design

This phase is completed with a preliminary design review (PDR). At the end of this phase the modules that will comprise the product, the interconnections among modules and the data structures are known. The PERT sizing method based on the beta distribution can now be used and so can SMM, and QSM SCS. Because detailed data flow diagrams have been developed BANG can be applied. With the functional description of each module a detailed description of all algorithms will have been defined, so ASSET-R, with its requirement for detailed knowledge of engineering formulas and equations used (to derive operator/operand counts), can be also be applied during this phase. ASSET-R has been included here, rather than in the detailed design phase, because the engineering formulae and equations are likely to be important enough to be defined earlier than many of the other algorithms. These engineering formulae and equations may in fact have been important enough to have been defined during requirements. If Curve Fitting, ESD and SSA are applied at the module level they can now be used.

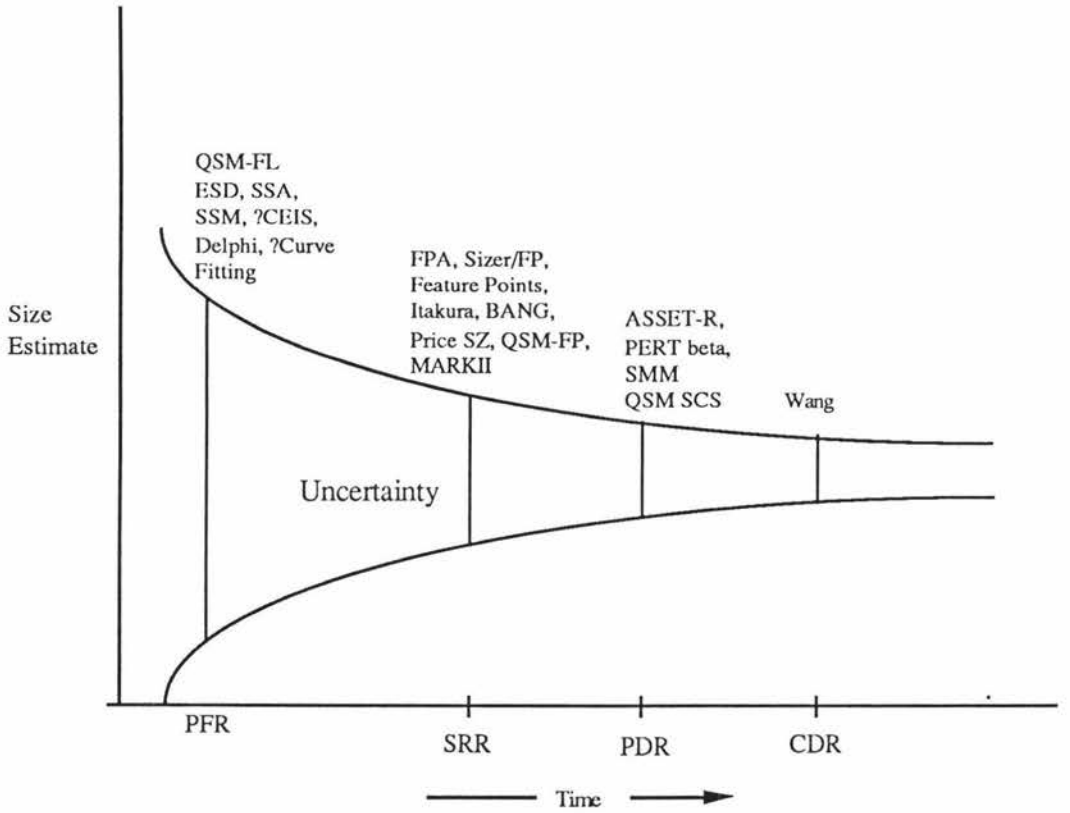


Figure 3.15 Relationship of Size Estimation Approaches to Phases
(modified from [DACS87 p.3-8)

3.4.4 Detailed Design

Detailed design is completed with a critical design review (CDR). Wang's data-structure-oriented approach was described as being applicable "at the end of the design stage [WANG84 p.141] and can be used at or after CDR.

Software Size Estimation Approach	Phase	Minimum Information Required
PERT normal	PFR	system type, general
Delphi	PFR	system type, general
SSA	PFR	functions
QSM Fuzzy Logic	PFR	system type
ESD	PFR	functions
Curve Fitting	PFR	functions
CEIS	?PFR	task attributes
Feature points	SRR	as FPA
QSM FP	SRR	as FPA
Sizer/FP	SRR	as FPA
Price SZ	SRR	ouputs, inputs, system states, tables, analogs, functional bulkiness
FPA	SRR	inputs, outputs, files, interfaces, inquiries
BANG	SRR	functional primitives, input and output data elements
[ITAK82]	SRR	reports
MarkII	SRR	files, inputs, outputs
PERT beta	PDR	modules
SMM	PDR	modules, variable counts
QSM SCS	PDR	reports, programs, subsystems, screens, files
ASSET-R	PDR	as FPA plus engineering formulae and equations

Table 3.6 Summary of Major Size Estimation Approaches and Minimum Information Required for their Use

Note: Knowledge of application development language to be used is assumed in the above table where appropriate.

3.5 OTHER CONSIDERATIONS

3.5.1 Static and Adaptive Approaches to Sizing

Many of the approaches presented in this chapter can be described as static, for example FPA and most of its variants. Dynamic or adaptive methods are those that continually add appropriate new and updated data to the completed project data base and allow the user to access this new data (for example CEIS and QSM SCS) for future estimates. MarkII with its tuneable coefficients can be regarded as an adaptive approach but its single component type may not lead to accurate size estimates. System components, whether modules or screens and reports, are dependent on the technology used and will change over time with the technology. FPA has been successful because it is based on system requirements components; however, these components are expressed in terms of a software development technology that may soon become obsolete.

3.5.2 Purpose for which Sizing Method is Suitable

There are three major purposes for which software sizing exercises are carried out. The first is for an estimate of the size of the software that will actually be written so that good software cost and schedule estimates can be achieved. The second is for an estimate of the size of the completed software product that will be delivered to a customer. These two sizes may be quite different because of modification and reuse of code, JCL, included code, etc. All of the software size estimation approaches described in this chapter that can provide estimates in LOC can be used for these two purposes. The third purpose is for software productivity studies. The only approaches that can be used for this latter purpose need to be applicable across languages. Only those methods that claim some degree of language independence (i.e. the FPA-based approaches) can be used for this purpose.

3.6 CONCLUSIONS

This chapter has introduced a new structural classification of software estimation methods (summarized in Figure 3.1) which complements, and at its leaves includes the more traditional approach-based classification. Not only is the new structural classification more formal than the traditional one, it also more clearly demonstrates the fundamental concepts within the different software estimation methods, particularly those which involve component partitioning.

Three other important, and complementary, classification aspects are also considered, namely a phase-based classification related to the phases of the software development method used, a brief consideration of static versus adaptive sizing methods and a classification according to the purpose of the estimate. The phase-based classification is a useful basis for the continuing refinement of a size estimate as development proceeds. Twenty two different methods or approaches have been classified and described. Most of the individual methods have been summarized in diagrammatic form by the author for greater ease of understanding. The individual descriptions of sizing methods identify the application areas for which they have been designed or to which they apply. These range from business applications to embedded systems.

The particular methods which have been analysed in more detail than the others are Albrecht's FPA and Symons' MarkII. The sections devoted to these methods contain some new critical analysis.

The concept of partial size estimates is introduced and used to relate adjustment factors to the purposes of a sizing exercise. A potential inconsistency between the adjustment mechanism in FPA and the concept of component sizing is uncovered.

Size estimation can only be as good as the level and completeness of knowledge of the proposed system; incomplete knowledge of requirements can only result in incomplete size estimates. It is also necessary for the user of any sizing method to be quite clear about the purpose and timing of the sizing exercise in relation to the capabilities and options of the available methods in order to choose a method suited to a particular sizing problem.

CHAPTER 4

A GENERIC MODEL FOR SOFTWARE SIZE ESTIMATION BASED ON COMPONENT PARTITIONING

The size estimation approaches discussed in Chapter 3 take a number of different views of the structure of the software whose size they attempt to estimate. Those methods that estimate individual component sizes, followed by summation, and possibly an overall system adjustment, appear to be the most promising approaches to more objective and accurate size estimation. FPA and most of its variants fall into this group. Although FPA has been criticized and many of the criticisms are valid, the approach is a useful one which is widely used especially for size estimation of business systems, and (with extensions) for real-time systems. In response to justifiable criticisms, there is a need for new FPA-like methods to be developed both for size estimation and also for productivity studies. Few methods, other than those in the FPA family, cater for purposes other than that of software product size estimation.

Because of the wide range of applications, development methods, and technologies, a more general FPA-like approach is required rather than a single method tailored to just one development method and technology or even a fixed set of methods covering several application categories. Such a generic approach, from which sizing model instances tailored to particular application categories and development technologies can be derived, has been developed by the author and is described in this chapter. This approach arose from an attempt to generalize FPA to give more objective estimates of the comparative sizes of component type instances, and to overcome the criticisms of FPA [VERN87, VERN89, SYMO88] described in 2.5.1 and those of MarkII discussed in 2.5.2 (iii).

4.1 CONTEXT OF THE MODEL

Although this thesis is not concerned with software costing, it is useful to put the software sizing process, with which it is concerned, into a wider software economics context. Figure 4.1, reproduced from [VERN87], presents a general model of the software sizing and costing processes. The following explanation of Figure 4.1 is

based on [VERN87]. Within the model are three parts A, B and C, with the following characteristics:

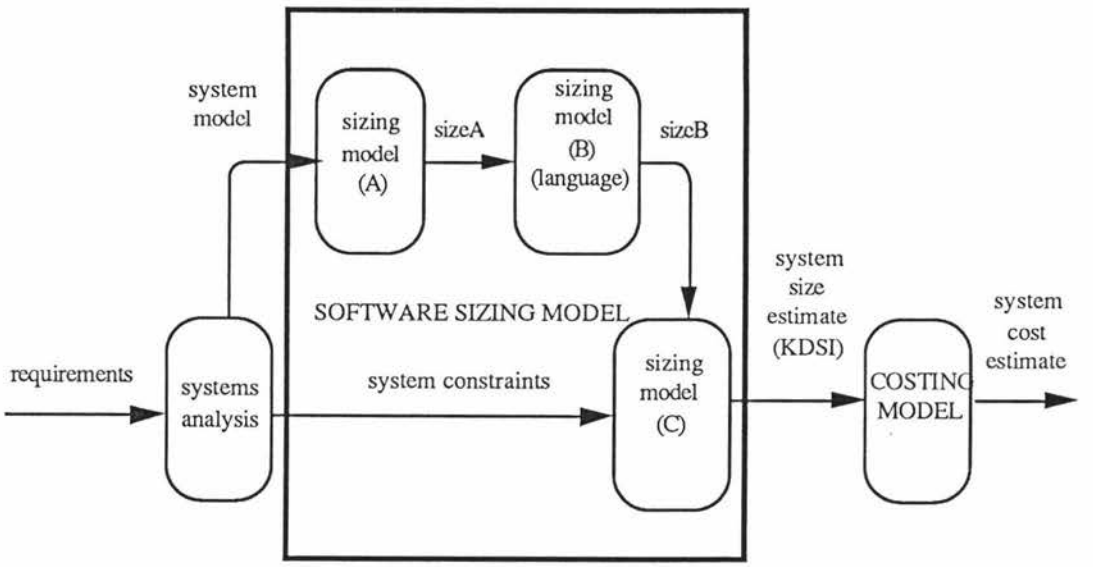


Figure 4.1 Software Sizing and Costing Model

Part A: SizeA is intended to be as close to the inherent system size, or information processing size (in the Albrecht [ALBR79] and Symons sense [SYMO88]) as possible, and as free as possible from implementation 'accidents' (in the Brooks [BROO87] sense. It is the 'size of the job to be done' of 1.3.1. A major goal for part A of the sizing model is objectivity. As was noted in 2.8 there is a need for improved language-independent metrics that can be calculated automatically from entries in a data or system dictionary. There is no good reason why SizeA should not be computed directly from an automated system model as a by-product of the system definition process. A critical issue is the unit of measure for SizeA. Candidates include specification token counts, LOC in a standard language, or a less language-dependent measure, such as function points [ALBR79, ALBR83], MarkII function points [SYMO88], System BANG [DEMA82, DEMA84] or a similar composite metric derived from a metric vector. SizeA is independent of the programming language used, and of other system constraints, but not necessarily completely independent of the software development technology employed, including the analysis and specification concepts.

Part B: This part of the sizing model, which may use one or more language expansion ratios (frequently expressed in LOC per FP) has the role of mapping the language-independent SizeA to the language-dependent SizeB. SizeB can be thought of as being the size of a 'neutral' source coding of the system in the chosen language. Typically this will be concerned primarily with functional aspects of the system rather than, for example, system utilities and housekeeping. Where several languages are used, or where the mapping is different for clearly definable parts of the system, SizeB is a weighted sum of components.

Part C: The third part of the sizing model aims to apply appropriate adjustment factors, based on system constraints, to SizeB, or to parts of SizeB, to obtain a realistic final size in KDSI. The frequently quoted work of Weinberg and Schulman [WEIN74] suggests that such adjustment factors can, in special cases, be much greater than the ±35% limit of FPA. In most systems, large adjustments would, however, only apply to parts of the system, for example, the user interface or data communications.

The main difficulty, or challenge, presented by the above model, is the clear definition of Parts A, B and C in a manner that allows each to be measured separately in a practical manner. In many development situations, only SizeC, the final source code, or its equivalent, is directly measurable. In some other situations, including the detailed examples analysed in this thesis, a close approximation to SizeB is measurable and an estimate of SizeA is also obtainable.

SizeA is essentially a functional specification size. Given a suitable formal specification language, or set of system description dictionary entries, there is no reason in principle why SizeA should not be measurable. For example, a suitable metrication of 'upper CASE'[1] tools could provide measures of specification size for environments using those tools. These measures though not in any way absolute, could provide useful information for comparisons with other developments using those same tools. However, a means of measuring SizeA directly is not yet commonly available, and is

[1] Upper CASE is a term used to describe computer assisted support for the early phases, or upper levels, of development.

not the subject of this thesis, being a substantial separate, though related, area of research. Nevertheless, as outlined in the discussion in 4.3 and 4.4, the general sizing model presented below caters for the case where SizeA is directly measurable.

SizeB is described as a 'neutral' implementation of the functional specifications in a particular language, or implementation system, without the system constraints. To the extent that functional modules, or components, can be separated and distinguished from system constraint components, SizeB can be considered a *partial size* which counts only the functional components. This concept is used in the sizing model developed in this chapter.

To understand the significance of Part C, it is necessary to be aware of what is meant in this context by system constraints. System constraints include requirements not in the 'what' category (i.e. not functional requirements) which can be consolidated into size drivers, for example response time requirements, aspects of system decentralization or distribution and a number of other housekeeping matters such as data base reorganization, system administration and accounting, security, backup and recovery. In general terms, the system constraints are related to 'how' the system is implemented, rather than to 'what' it does.

4.2 INTRODUCTION TO THE MODEL

A generic model for software sizing is presented in the form of a set of steps which are followed to build an instance of such a model. Though the steps are presented as a sequence (strictly a network) of activities, there is likely to be considerable iteration in their actual use. Following the presentation and explanation of the model in 4.3, a number of issues in relation to the model are discussed in greater detail in later sections.

The model is generic in that it is intended to cover sizing problems which can be solved by *system decomposition* and the subsequent *summing of component size estimates.* An important aspect of sizing by decomposition into components and summation of the component sizes is that (as noted in 4.1) it allows **partial sizing**, based on some, rather than all, of the components of a system. Partial sizing may be useful in that it allows different aspects of a system to be considered separately. In many cases, for example, functional aspects of a system are separable from a number of implementation-

dependent aspects by a suitable component decomposition. In many organizations, the functional aspects are handled by application programmers whereas systems aspects are handled by systems programmers.

The model is also general in the sense that provision is made both for product size estimation at various stages during development and also for size comparisons for productivity studies. As noted in Chapter 1, the purposes of these uses of software sizing are different and they tend to require different software size measures. LOC, or their equivalent, are most frequently the goal of product size estimation. Productivity studies, on the other hand, may use function points, or their equivalent, and calibration across different software development technologies is extremely important. In terms of the framework of 4.1 the model can be used for

(i) estimation of SizeA
(ii) estimation, or after-the-fact measurement of a (partial) SizeB
(iii) estimation, or after-the-fact measurement, of SizeC.

In fact, if and when suitable measures of SizeA become available, the model should apply there also for measurement and for estimation. Which partial or total size estimates or measurements are relevant depends on the purposes for which the measurements are made, which are also often related to the development phase at which they are required.

4.3 A COMPONENT AND SYSTEM SIZING MODEL

Figure 4.2 illustrates the steps in the generic sizing model presented in this dissertation. Each of these steps is discussed in detail in following sections. Other related aspects of the model are discussed in later sections of this chapter (4.3.1 - 4.3.7) and the relationship of the generic model to the Bailey and Basili [BAIL81] meta-model is discussed in Appendix D.

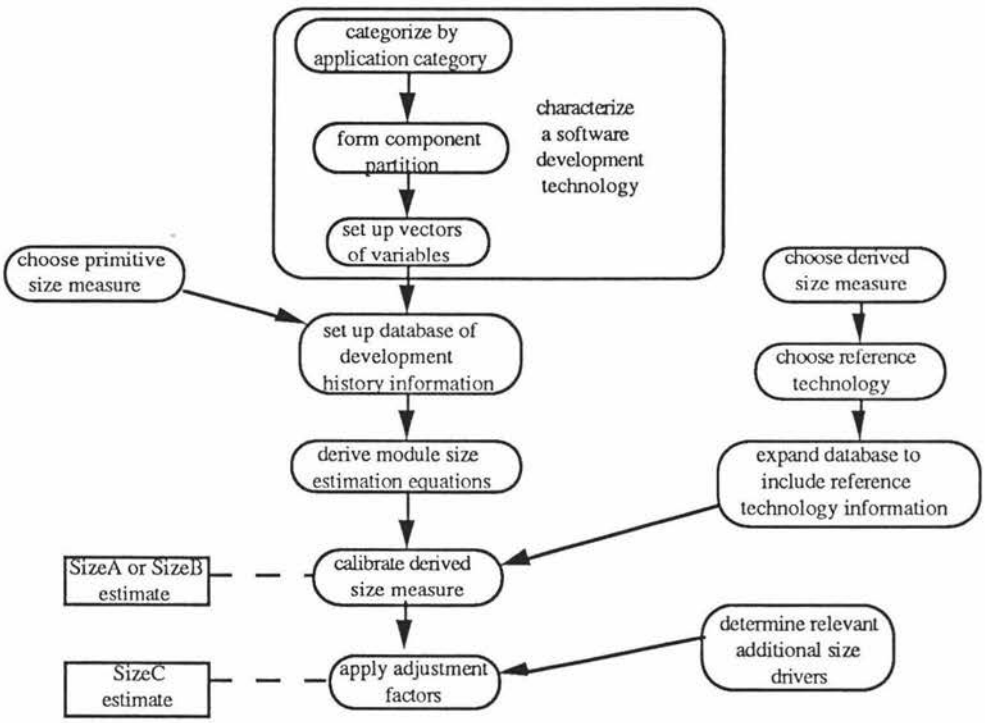


Figure 4.2 Generic System Sizing Model

4.3.1 Categorize by Application Type and Software Development Technology

An application category is a class of functionally and/or structurally similar systems. Traditionally, such categories have been very broad, often dividing all systems into just three categories, business, real-time and embedded. Boehm identified three categories which he termed organic, semi-detached and embedded [BOEH81] while QSM - Fuzzy Logic identifies eleven [PUTN87] (including microcode/firmware, real-time, avionics, system software, command and control, telecom and message switching, scientific, process control, mixed applications, unknown applications and business software), and ASSET-R three [REIF87] (data processing, scientific and real-time). Generally, those sizing models that give a choice of application category have identified a small number of categories whose characteristics and size drivers are sufficiently different to warrant separate treatment. However, as applications vary widely, a small number of categories may not be sufficient to characterize a wide range of applications, particularly since the

variability within each category should not be too great if good size prediction is to be achieved. It is also possible that some applications may fall between categories.

Though this activity is called "characterize by application type", this does not imply that it is necessary for the division into all application categories or types to be done at one time. It can equally well be done incrementally, one category of interest at a time. In some development environments there may only be one application category.

Characterization by application type is not in itself a well-defined process. It is, however, a necessary first step in the characterization of a **software development technology** which, in the context of this sizing model, is more precisely described by

(i) a set $\{phase_p\}$ of development phases, of interest as sizing milestones; for example, the case study in Chapter 5 uses the set {Phase1, Phase2}

(ii) a set $\{P(phase_p)\}$ of specific partitions into component types, characteristic of the application category, development environment and development phase; for example, the case study in Chapter 5 uses a Phase1 partition {relations, menus, screens, reports} and a Phase2 partition {relations, menus, screens, reports, updates}

(iii) for each development phase, and for each component type, a vector $v(P(phase_p))$ of object types and component characteristics typical of that component type at that development phase, and likely to influence its size; for example, the case study in Chapter 5 uses the vector (1, Number of relations, Number of data elements, Complexity of logic)[1] for components of type screen within the Phase2 partition.

In practice the formulation of these specific descriptors of a software development technology is the result of a complex classification process which is influenced by the application category, the development methodology (including its life cycle model) and the language, or more generally the software engineering environment, used. Since these major influences depend primarily on the application category, it has been highlighted in Figure 4.2.

[1] the entry 1 in the vector is for a constant term in an estimation equation, see Figure 4.3.5.

There are some difficulties in describing classification processes such as the above in general terms. The analysis of specific cases in Chapters 5 and 6 should help to further clarify many of the concepts.

The class of software development technologies, as used here, is potentially much larger than that of application categories, there often being many commonly used software development technologies for one application category. In particular this thesis examines four different software development technologies for the application category of data-centred business systems.

The question of the division of broad sets of systems and components into classes is a very difficult one, whether the classes are application categories, software development technologies or component types. Very few specific guide-lines can be given here as to when a new class should be created. This will depend on the sizing requirements of the user and is investigated further in an example at a component level in Chapter 5. Typically new classes will be considered when an existing class becomes too diverse, too many outliers with similar characteristics are present, and too many differences requiring *ad hoc* or other adjustment occur. A general guide-line might be that a class should be sufficiently coherent that estimates within it can be made within 25% of actual 75% of the time [CONT86], some defined maximum mean relative error, or mean relative root mean squared error[1] is not exceeded, or such other target accuracy that the user organization sets is achieved. These are simple enough principles to state, but in practice some classes which exhibit greater than desirable variability may simply not be divisible into more coherent subclasses on any conceptually satisfactory basis. If any two component types, or other classes, turn out to have similar estimation equations, consideration should be given to combining them.

There is however a trade-off between accuracy of estimation and breadth of applicability of a classification. For example, software development technologies, in the sense defined here, may be comparatively narrow and specific, as are those of Chapters 5 and 6, which satisfy quite stringent estimation criteria. However, a broad technology classification, such as that to which FPA has commonly been applied, may give rather more mixed results.

[1] relevant statistical evaluation criteria are summarized in 5.5.2

4.3.2 Form Component Partitions and Set up Candidate Variable Vectors

Depending on the scale of a system, the development phase, and what information is available, more or less may be known about how many components a system has and what kinds of components they are. The components may be at one of a number of levels, from sub-projects, or subsystems, down to individual procedures. They may have different names, such as modules, tasks, functions, processes, programs, objects, entities, relations, etc. Data and procedural specifications may be combined within a component, or they may be separated into different components. There may be a single component type or there may be a number of different classes. Components may refer to specification objects, to design objects, or to implementation objects. The discussion below is intended to cover most of these cases, except those where the actual number of components, or at least an estimate of the number, is not known. In such a case, of course, estimation by summing components is not possible. The model can readily be adapted to apply sampling from a known or assumed component population.

Systems are often specified in terms of particular components or structures, for example in a specification based on data flow concepts the components of interest may be:

data flows
data stores
processes
external entities
data structures
data elements

to which might be added in practice:

screen/window descriptions
report descriptions.

Systems are usually implemented in modules (programs, procedures, definition blocks, include blocks, or any distinguishable component) which are often of different types.

For example, a business system implemented using a 4GL or CASE environment may consist of:

file, entity or relation definitions
screen, and/or window, definitions
report definitions
procedures
system administration definitions.

In a Modula-2 environment, one might have, at the design stage

definition modules
implementation modules
procedures.

The term **component** may in various contexts include modules, structures, objects or other distinguishable and separately countable units. Typically, different component types have quite different purposes and their sizes are determined either by different variables, or the determining variables have different importance, or a combination of both. Note that the components of interest at the requirements phase may be different (at least in part) from those at the specifications phase, which may again be different from those at design, and possibly again at implementation. It is however expected that the components at all phases are closely related developments or refinements of the earlier phases. It is difficult to see how the integrity of a development model can be preserved if this is not the case.

A **component partition** refers to a division of the components of a system at any phase or convenient milestone in its life cycle into a mutually exclusive set of **component types**, covering all of the aspects of a system which are of interest at that phase of its development. Note that this need not necessarily include all parts of a system. It may, for example only cover the application data and its processing, i.e. functional aspects (SizeA and SizeB), and omit any overall systems consideration, such as system administration, utility or distribution functions (i.e.. SizeC minus SizeB). The FPA-based software sizing models and Price SZ (which are discussed in chapter 3), apply a greater or fewer number of adjustment factors to a partial size measure to get the overall system size of interest. Section 3.2.3 discusses adjustment factors in detail.

For each component type, a **vector of candidate variables** is set up. These variables include those from which the size of a component of that type can be estimated. The term candidate is used in this context because not all the variables in the vector may be used for a particular purpose. For a report component, for example, the vector of candidate variables might include:

number of entities or relations involved
number of data elements (fields)
number of sort fields
number of selection fields
complexity of selection logic
number of control breaks
number of report line types
complexity of report calculation.

However, in some circumstances, not all of these may be known and, whether known or not, not all may be significant.

Figure 4.3 illustrates an example of categorization by application type, formation of component partitions and the setting up of candidate variable vectors. As noted in 4.3.5, the first entry in a vector of variables is, by convention, 1.

For each component partition and, within this, for each component type, an **availability profile** may be constructed showing at what stages of the development process, and for what purposes, the particular component partition and candidate variable vectors are relevant and when the information that they use is available.

The criteria for setting up, or selecting, component partitions, will depend on the purposes for which a sizing exercise is done. For example, technology productivity studies will often use component partitions related as much as possible to the intrinsic or specification size of a job rather than using categories which are more dependent on how the system is designed and implemented. For software cost estimation the interest is often in a measure of the system's total representation size. Thus, the choice of a suitable component partition will depend on the purpose for which it is being used. Some partitions may well be satisfactory for several sizing purposes, however.

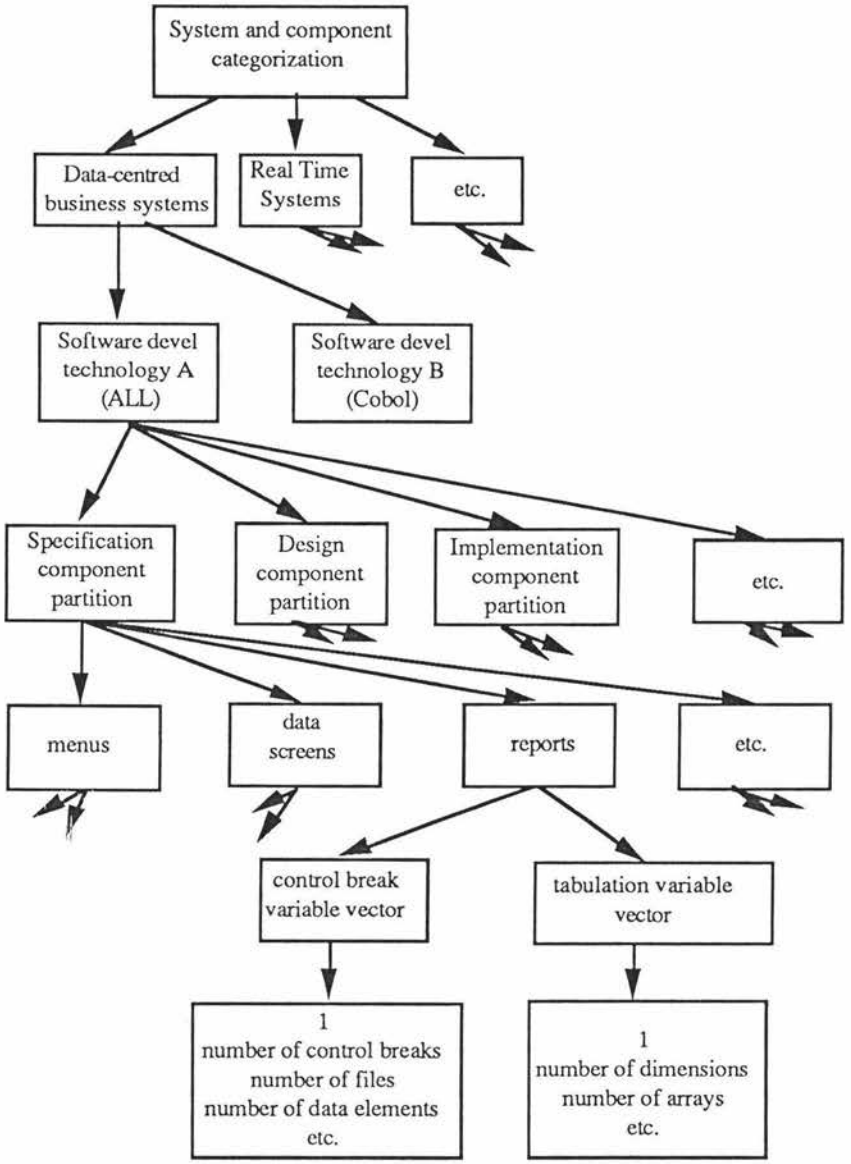


Figure 4.3 An Illustration of System Decomposition

In terms of the general sizing and costing model developed in this chapter, the following distinction between component partitions and variable vectors related to SizeA in comparison with those related to SizeB and SizeC, is important. SizeA may be difficult to measure directly, and some *arbitrary or derived measure*, such as function points may therefore be used for SizeA, while SizeB and SizeC can be measured *directly*

using a *primitive* measure, such as LOC. The distinction between primitive and derived, or composite, measures is examined in more detail in 2.7.

For clarity and precision, in the following discussion of the model, a subscripting convention will be adopted as follows:

i will relate to the component type, for example in a division of functional components into files, screens, reports and procedures, a value of 3 for *i* would indicate a report type,

j will relate to the vector element concerned, for example a value of 2 for *j* might indicated the 'number of data elements' entry in the vector example shown in the text above,

k will indicate the index of the particular component, for example a value of 20 for *k* would indicate the 20th report, if one were considering report components.

Thus $c_{3,2,20}$, in the context of the examples used, would indicate the number of data elements in the 20th report.

4.3.3 Choose Primitive Size Measures

The term primitive could be replaced by *basic,* and both of these terms could be further qualified by the adjective *common.* The size measures are primitive, or basic, in that they are low level or elementary measures relating naturally to basic implementation structures, such as statements, or LOC. They are common in the sense that they apply to all, or most, implementation structures and to all, or most, implementation languages. In terms of the general sizing and costing model of 4.1 and figure 4.1, they are normally usable as measures of SizeB and SizeC.

Though, as noted in Chapter 2, the concept of a single size for a system has many unresolved problems, in practice systems cannot effectively be compared, nor can schedule and cost estimates be subsequently developed for them, without using at least one single common measure of size, such as LOC. There may be good reasons to use more than one such measure. These measures might include:

(a) *LOC, source instructions, or delivered source instructions.* These tend to be used interchangeably, and there is some reason for doing so. For example, in the case study of [VERN88] part of the source 'code' was in the form of lines of non-procedural information (not organized in any statement format) filled in on specification screens, while part was in the form of procedural statements in a one-per-line format. Though the procedural statements were clearly source instructions, and the non-procedural instructions were not, they could be equated for sizing purposes as LOC. LOC, as noted earlier (Chapter 1), are essential as input to most algorithmic costing and scheduling models. LOC are a small, but not the smallest, unit useful in this context.

(b) *Token counts.* These are at a lower level again than LOC and may have validity in some situations, for example in some CASE tools, where no suitable equivalent to LOC seems to exist (see 2.2.3). Token counts are the smallest useful unit in this context.

For the primitive measures selected, it is necessary to have clear and consistent definitions, expressed if possible algorithmically, which can be used for automatic counting.

4.3.4 Set up Development History Data Base

The term data base is used here in a general sense. It may refer to a conventional data base using some data base management system, or it may refer to a set of spreadsheet tables. The data base will contain information about the software development technologies of interest, including their component types and candidate variable vectors. Initial estimation will be based on this data. As each new project or increment is completed its data should be added to the history data for its technology. The estimators can be recomputed, either whenever any new data is added to the data base, or when there is evidence that the estimators may have changed significantly; the latter course of action will often be desirable for comparative purposes, see 4.3.6 below. Tools should be used wherever possible, or development tools instrumented appropriately, to obtain the necessary counts and enter them into a data base automatically.

4.3.5 Derive Component Size Estimation Equations or Algorithms

Many possible algorithms or estimation procedures could be used for this step, including ranking (Curve Fitting [DACS87], SSM [BOZO86]), sampling (SMM [BRIT85]), and reference to a data base for selection and comparison with similar historical cases (QSM SCS [PUTN87], ESD, SSA [DACS87]), for example.

In order to be specific, however, a particular estimation method is used here, namely a straightforward application of multiple linear regression to data in the history data base to derive estimation equations. Estimation will be done on a component by component basis. It has been assumed in the following that the dependent variable is the primitive common size measure LOC but in principle, it could be any measure of size, or even of value. If appropriate, tokens or other suitable measures can be used instead, or in addition to LOC. The independent variables to be used are those occurring in a specific candidate vector of variables for a component. If there are several such vectors for a component, then there will be several estimation equations to be used at different stages of the development depending on when the information for the vector entries becomes available.

Thus, for the kth component instance of component type C_i, its size estimate L_{ik}* in LOC is

$$L_{ik}^* = \sum_j w_{ij} c_{ijk}$$

where w_{ij} is the regression coefficient , or more generally weight, of the jth variable, and

c_{ijk} is the value, normally a count, of the jth variable.

By convention, the first variable in any vector, c_{i1k}, has a value of 1 and assumes the role of the constant term in any regression equation. In the rare case (such as Albrecht's function point analysis) where there is no constant term, $w_{i1} = 0$. Thus, in all cases,

$$w_{i1} c_{i1k} = w_{i1}.$$

This convention avoids making a special case of the constant term in regression equations.

The choice of which variables to include in the estimation equation will normally depend on which variables in a vector are available (typically related to the development phase) and also which variables the regression or other algorithm finds to be significant.

The estimated size in LOC, L*, of a system S, made up of components from a component partition C, whose individual sizes are estimated as above, is

$$s^*(S,L,C) = \sum_i \sum_k L_{ik}^*.$$

It is important to stress again that estimation based on regression analysis with LOC as the dependent variable, though a powerful and successful technique, is only one of many possible approaches that could be adopted within the framework of the model.

4.3.6 Choose Derived Size Measure and Reference Technology

This step is necessary if productivity studies are to be conducted, either across different software development technologies, or as a software development technology changes and develops. Productivity is about how efficiently a job is done. To measure it we need to know both the size of the job itself and the eventual cost of doing it. Since the latter depends strongly on the size of the implementation, in the context of this model the following is needed:

(i) a measure of the inherent size of the job (SizeA), - typically function points, a derived measure, and
(ii) a measure of the size of the software product (SizeB or SizeC, depending on what aspects of productivity are being measured) - typically LOC, a primitive measure.

The derived measure, or measures if several are used, will typically be function points, or some development or generalization of them, such as the extended function points used in ASSET-R [REIF87], or the feature points of SPQR [DACS87]. Such measures

are grouped under the term **FPA-like** and their unit of measurement is called the function point. Function points are at a higher level than LOC and, though not completely independent of the software technology employed, are closer to 'the size of the problem', the size of the system as described in the requirements, or the inherent size of the job to be done (SizeA of Figure 4.1), than are LOC. LOC relate more to implementation considerations, such as the language used (SizeB and SizeC of Figure 4.1).

It is important to reiterate that function points are not in the same category as either LOC or token counts. Specifically, function points are a complex, or composite (derived) measure, whereas LOC and tokens are simple or primitive measures. Function points cannot normally be calculated directly from source code, or its equivalent, whereas LOC or tokens can be so calculated. Function points are calculated, or more precisely *estimated*, using techniques essentially similar to those of this model. This is discussed in greater detail in 2.7.

For technology productivity studies, commonly used or previously used categories should be included if comparisons with previous systems measured using them are required. For example, for comparison with many existing systems which have used Albrecht's function point analysis [ALBR79, ALBR83], the component partition

logical internal file types
inputs
outputs
inquiries
interface file types

is necessary, together with corresponding candidate variable vectors, such as

number of file types
number of record types
number of data elements.

This may not be the natural or the best, partition to use. In practice, however, it is a partition that can, using suitable conventions, be imposed on most business systems (in addition to the preferred partitions) where this is necessary for calibration purposes.

4.3.7 Calibrate Derived Size Measure

In the case where an FPA-like measure is used, for comparative purposes or for technology productivity studies, calibration will be necessary to ensure a consistent unit size for the (derived) measure for the purposes for which it is being used. Calibration implies the use of a reference software development technology, more simply called a **reference technology**, in which well-established, and sufficiently reliable, function point values have been obtained in the past. The reference technology selected must also be applicable to the newer systems which use the FPA-like measure being calibrated.

As stated in 4.3.1 the term **technology** in this context refers to a particular component partition, together with a particular variable vector for each component, to which must be added, for calibration purposes, a fixed regression equation for each component. Often, these will correspond to a particular method of specifying or developing software for which these components and counts are the natural ones. In relation to the software sizing and costing model of 4.1, the term technology as used here usually applies to subparts A and B of the sizing part of that model, though for some total sizing purposes, and also in situations involving some software development technologies, where it is not possible to separate out the system constraints, some or all of the system constraints might also be included, thus including some or all of SizeC in the technology. Whether or not the effects of system constraints, or of some of them, are measured separately is a matter of separability and of user choice.

Note that, in this context, the estimation equations should be fixed, not adaptive as optionally suggested in 4.2.4 above. Adaptive equations would imply a 'drift' in the software development technology. Until there is a significant change, it is unlikely to be worth refitting the estimation equations, and when this is done recalibration against the reference technology will also be necessary.

The calibration process, using a reference technology A (typically Albrecht's FPA) for calibration of an FPA-like measure in a new technology B, requires the following size measures for a number $n > 0$ of systems S_m:

(a) LOC or other primitive measure for development using technology B, $s(S_m, L, B)$

(b) function points, or similar, in a reference technology A, $s(S_m, F, A)$

Calibration of the FPA-like measure for technology B involves the determination of the ratio b, which is the number of technology B LOC per reference technology A function point for the same set of systems S_m. If equal weights are given to each system S_m, b is given by

$$b = (1/n)\sum_m(s(S_m, L, B) / s(S_m, F, A)) .$$

To determine function points in technology B for both components and systems, it is merely a matter of dividing the LOC estimates in 4.3.5 by the calibration factor b. This may be done by scaling the weights (or regression coefficients) w_{ij} to new values w_{ij}/b. However it is probably better to divide the LOC estimates themselves by the scaling factor , thus bringing the role of the latter out into the open.

A simple example, Table 4.1 may clarify the calibration process.

	System 1	System 2	System 3	System 4
Size in new technology, B	20000	34000	4500	6000
Reference technology A Function Points	201	390	50	58
s(Sm, L, B)/s(Sm, F, A)	99.50	87.18	90.00	103.45
Σm	380.13			
b	95.03			

Table 4.1 Determination of Calibration Ratio, b

The question of the calibration of derived or composite measures is examined in a more general context in 4.4.

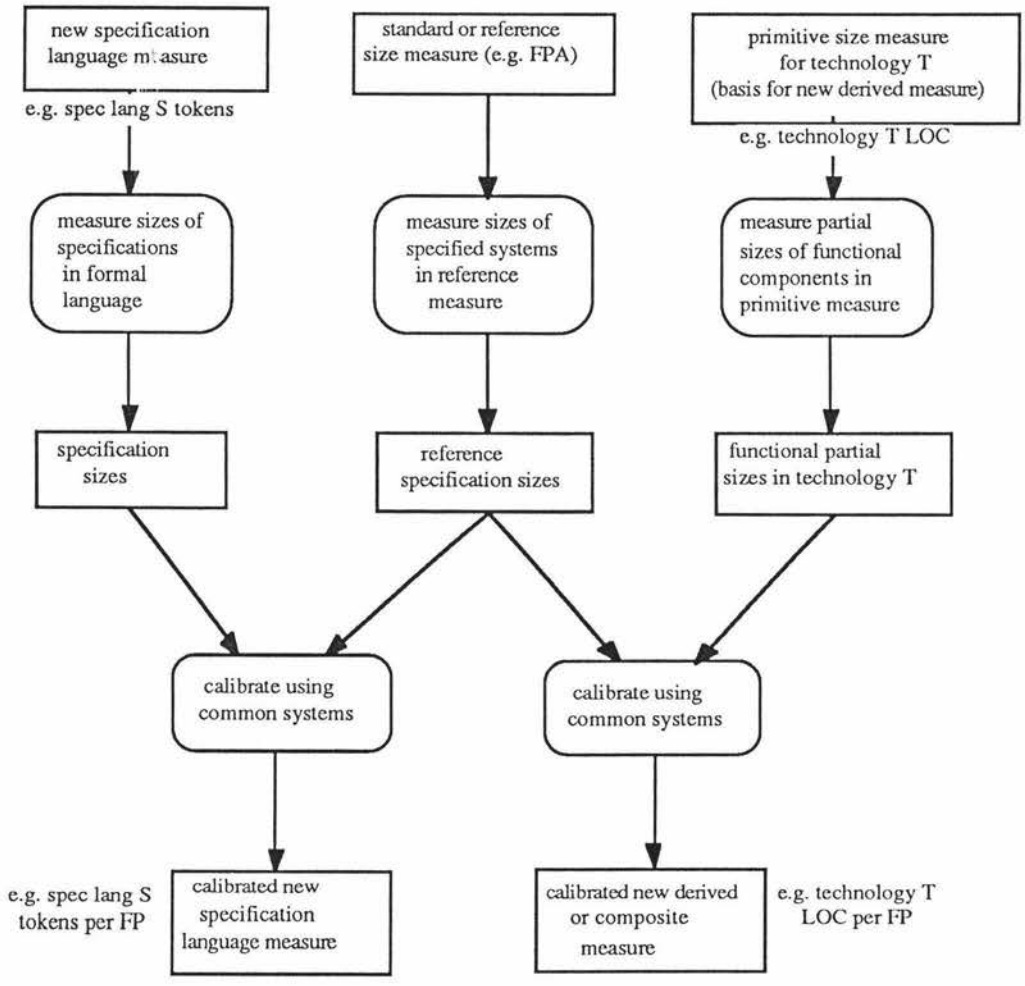


Figure 4.4 An Example of Calibration for New Size Measures for Productivity Comparisons

4.4 CALIBRATION PROCESS FOR NEW SOFTWARE MEASURES

Figure 4.4 gives an example of the calibration process for two new software measures (or estimates) in a more general context than in section 4.3.7. On the left, a new specification size measure has been formulated (SizeA), either arbitrarily (for example, 'by debate and trial') or using a primitive specification size measure alone (for example tokens in a specification language). This is then calibrated against an existing measure, such as function points, for comparison with existing size information. On the right, a derived measure from a (partial) SizeB in technology T LOC is calibrated using the

method described in section 4.3.7. It is clear from the symmetry of the diagram that the same principles apply in both cases and that the differences in calibration method are minor. For the new specification size measure on the left of Figure 4.4, one could obtain a calibration in terms of specification language tokens per reference technology function point. For the case on the right of Figure 4.4 one could obtain a calibration in terms of technology T primitive measure units per reference function point.

Ideally, one would like an independent measure of size at each major development milestone, and this may eventually become available with CASE tools. In such a case, the situation could be as illustrated in Figure 4.5, and the effects of specification-design expansion, and design-implementation expansion could be separated and measured more precisely. The stages of specification, design and implementation illustrated in Figure 4.5 are, of course, just examples. Any definite milestones at which independent size measures are obtainable could be used instead of them.

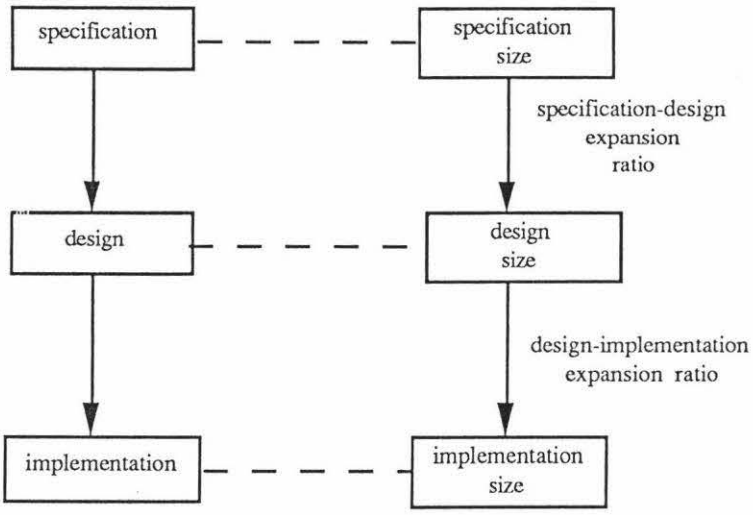


Figure 4.5 An Example of Independent Size Measures at Different Stages of Development

4.5 ADJUSTMENT FACTORS

4.5.1 The Role of Adjustment Factors

The dependence of adjustment factors, if and when they are required, on the purpose for which a particular (partial) size is required, is noted in 3.2.3. For the calibration of derived size metrics for technology productivity studies (where an approximation to the abstract size of the job to be done is usually required) adjustment factors will be of little or no interest. However, for product size estimation (which must take account of development methods, languages, and constraints) a number of adjustment factors may be important.

The use of adjustment factors in sizing models is reviewed in 3.2.3 where it is noted that the FPA approach of up or down adjustments to an average size is inconsistent with the concept of component size estimation, or at least unnatural in this context. For this reason it is rejected in favour of the approach described below.

The approach to adjustment factors adopted here is what is believed to be the more natural one of

(i) estimating component sizes as closely, and completely, as possible

(ii) summing the individual component sizes to obtain a (partial) system size

(iii) if necessary, adjusting the partial system size upward as appropriate to account for factors not included in the partial size obtained from the summed component sizes

Note that the term partial is used in a similar sense to the mathematical term subset. It may include all parts of the system that are of concern.

The key questions, then, concerning adjustment factors are:

(i) What factors of importance have been left out? What else needs to be taken into account?

(ii) Where in the system do the omissions, or shortfalls, occur?

(iii) What is their nature? Do some of them affect existing component types? Do some of them fall into additional component types not yet defined? Do some of them have a more or less constant effect, independent of system size? Are some dependent on the size of a part, or the whole, of a system?

These considerations lead to the conclusion that, while provision for adjustment factors should be made in any software sizing model that can be termed generic, few rules concerning them can be formulated and their use must remain highly dependent on the purpose of size estimation and the management environment in which it is used.

4.5.2 An Adjustment Factor Model

The treatment of adjustment factors in the sizing model presented here is illustrated in Figure 4.6.

The first step is to determine whether adjustments are necessary or not for present sizing purposes. If necessary, determine which adjustment factors are important in the current situation.

The second step is to classify the adjustment factors into five classes within two main groupings:

(i) **Module adjustment**, either new component types are required or only certain component types are affected.

(a) *Only some component types are affected by the factor.* Factors in this class increase the size of certain component types within the system but do not affect others, for example user transaction logging which, in a particular environment, may affect screens, but not relations, menus or reports. These factors should normally be included explicitly as estimation variables in the size estimation algorithms of the component types they effect.

(b) *New component types required.* Factors in this class can form separate measurable component types in the completed system and should, in general, be estimated separately, for example, the update component type of the Phase2 partition of 5.6.2.

(ii) **Size-dependent factors that affect the whole system.** These adjustment factors cannot be conveniently identified as increasing the size of any specific component types(s) but do increase the size of the system as a whole. They are often 'hidden' housekeeping functions, such as data base reorganization, systems accounting, security, backup and recovery, or other essential systems programming utilities. They may be:

(a) *Fixed size adjustment factors.* An additive adjustment is appropriate for this class.

(b) *System size-dependent adjustment factors.* Such factors are related to system size and show a greater increase in size in large systems than they do in smaller systems. A multiplicative adjustment is appropriate.

(c) *Other.* Expert judgement is likely to be needed for adjustments which do not fit conveniently into one of the other four classes.

The final step is to apply the appropriate adjustments. Note that in the estimation of any specific system the above classes of adjustment factors are not mutually exclusive, and any or all of them may be appropriate.

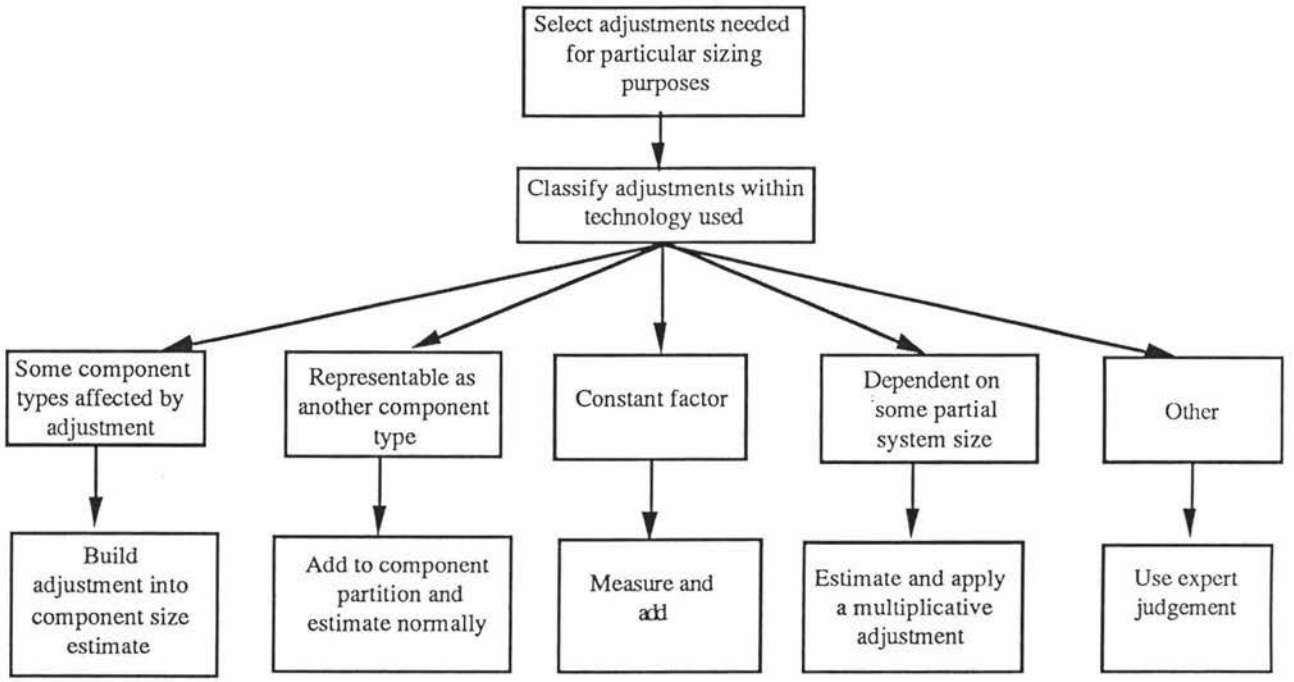


Figure 4.6 Adjustment Factor Model

4.5.3 The Place of Adjustment Factors in this Thesis

Although adjustment factors have been examined in some detail in the interests of completeness, they are not a central feature in the presentation of the model in this thesis. After classifying them and providing a framework for their use, if necessary, in the final step of the size estimation model, they are not considered or used again here. None of the estimates in Chapters 5 and 6 use adjustment factors, except in the calibration considerations of 5.8. Where Albrecht's FPA is used, only unadjusted function points are estimated.

4.6 CLASSIFICATION OF THE MODEL

It is interesting to review where the generic model described in this chapter corresponds to the structural classification of software size estimation models described earlier (Chapter 3). Figure 4.7 is a reproduction of this structural classification with the

addition of shading to show the parts of the classification subsumed in the model; the darker the shading the more relevant that class or sub-class is to the model. The model fits directly into the major class which estimates size by partitioning a system into components. The model is not at all concerned with attempting to size a system as an individual whole. Within the broad classification of *size estimates from partitioning a system into components*, it is possible to use the model in a number of different modes. The different modes enable variants of the model to be used in all the four sub-classes that fall within this classification with a resulting greater or lesser degree of accuracy in size prediction. Use of the model in different 'modes' within the four classes is described below.

4.6.1 Several Component Types

(i) Individual component sizing

The model fits directly into this group and this is its normal mode of operation. The model is different from most other members of this class in that it has no fixed set of component types and it makes explicit provision for size estimation during more than one phase of software development. This provides the facility for different size drivers with varying importance to be identified for component types at different phases. FPA has a fixed set of component types with fixed size drivers within component types. SSA has a wider set of standard component types from which the user can choose but the user must decide subjectively from the range and complexities of the data retrieved from a data base where the new module's size will fit. The Itakura and Takayanagi approach is in some ways similar to the model but is for one specific component type with subjective coefficients for the size driver variables. The model also has some similarities to System BANG with its open-ended set of functional primitive types. However System BANG divides the system into these primitives from a functional viewpoint whereas the component types delineated in the model not only have different functions but are also structurally different.

(ii) Size with average component size

The model can be used in a mode similar to the size estimation methods in this class. To do so the sizes of members of each specific component type will need to be investigated and an average size for components of that type computed. This has in fact has been done for one component type in an application of the model described in

Chapter 6. This was necessary as there were not enough examples of this component type to obtain a prediction equation. Using the model in this fashion will not normally give estimates as good as those resulting from the use of the model in its normal mode unless the sizes within a particular component type have little variability.

4.6.2 Single Component Type

(i) Individual component sizing

The model could be used in a single component type mode if this were required. To do this would require all component data to be put together and a single regression (or other) prediction equation to be developed from this data. It is possible that some of the variables in the component type estimation equations (for example, choices) would only be appropriate for some components and for the majority of component instance estimates the value for that variable would be zero. It is unlikely that the prediction equation would be limited to the three variables used in the MarkII equations but the model would be similar in some respects to MarkII.

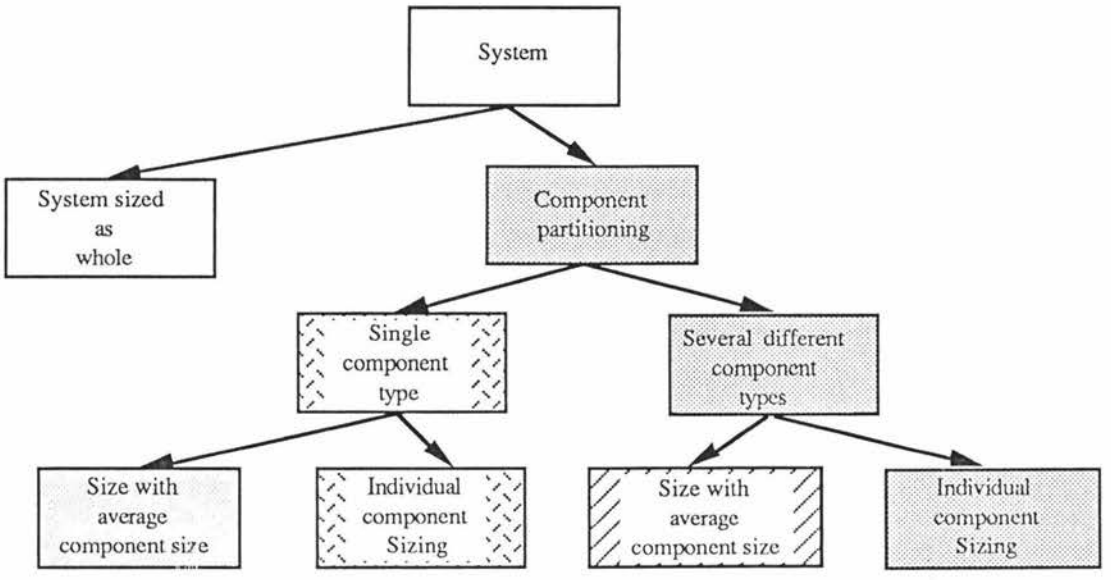


Figure 4.7 Sizing Method Types Subsumed by the Model

(ii) Size with average component size

Using the model in this mode is likely to give the poorest estimates. It reduces the model to one similar to that described by Aron [ARON69] (see section 3.2.2 (i) a) where the number of components in a system are counted and an average component size is used to estimate a total size for a system.

4.7 SUMMARY

The generic size estimation model described in this chapter is placed in the context of a more general model of the whole software sizing, costing and scheduling process, the SizeA, SizeB, SizeC model which separates language-dependent aspects (SizeB) and system constraint aspects (SizeC) from each other and from functional specification aspects (SizeA). The size estimation approach described here builds on the strengths of FPA-like methods, generalizes them, and overcomes most of the problems described in 2.5.1(iv) and 2.5.2 (iii).

Any group of applications that are sufficiently similar in their sizing characteristics can be regarded as an application category. Within each application category of interest, one or more software development technologies are delineated by defining suitable development phases, component partitions and vectors of object types and component characteristics typical of that component at that phase of development. Different component partitions may be of interest with different application types. The component partitions, and their vectors, will depend on the application category, development methodology, programming language and software engineering environment, stage of development, and the purpose of the measurement or estimation. This approach overcomes problems that may occur with a small number of fixed application types which may not fit the characteristics of a particular development environment or technology.

As described in detail in 4.3.3 and 4.3.6 the model allows any well-defined primitive or derived metric to be used thus better meeting the different purposes for which a size estimation and/or measurement model is required and the different styles of expression of emerging software technologies. The concept of partial size estimates and measurements is introduced and its relationship to different sizing purposes is emphasized.

The size estimation model described in this chapter is designed to adapt to changes in technology. The approach recognizes and uses the technology dependence of component types to get better 'SizeB' technology-dependent estimates instead of trying to ignore their dependence.

Although there will be changes in component partitions and vector weightings to meet technology changes, the model recognizes the importance of a stable reference measure platform for technology productivity studies. A method for calibrating an estimate of SizeA of a system in one technology, to other technologies (including later versions of a given technology), is described.

Within the model the choice of weights for components is no longer arbitrary but can reflect their comparative sizes (in the chosen metrics), within the technology currently being employed. The sizes of these different component types are determined from different variables, or with different estimation coefficients as appropriate using a history data base.

Since such a point has been made of technology-independence in some promotions of FPA (though it is not in fact completely technology-independent), it is important to note that a choice of component weights for an instance of the proposed model could be based on a measure of 'function value delivered to the user' if and when suitable metrics for this, or some other suitable measure of utility, are devised.

With the use of the approach described in this chapter the relative size of each component instance will be reflected more closely in the size of the estimate produced. This results from each component instance being estimated individually from counts of its associated objects in the development phases of interest; thus within a component class, a continuous range of sizes is possible, not just a small fixed set of sizes such as low, medium and high. This range of size estimates for component instances overcomes the criticism of FPA's oversimplified classification of component type complexity levels.

Because the model allows the use of more objective counting methods that may be automated the subjectivity inherent in traditional function point counts should be reduced.

The role of adjustment factors is analysed in relation to the purpose and scope of a sizing exercise. A new model for the identification, classification and application of adjustment factors is presented.

The generic nature of the model is emphasized throughout, especially in 4.6 where it is shown to subsume all other methods based on component partitioning.

The application of the general approach, described above, is illustrated in Chapter 5 where it is applied to a large set of related administrative applications developed using a particular software technology. It is applied again in Chapter 6 to a case in which a single standard application is developed using three different software technologies.

CHAPTER 5

APPLICATION OF THE SIZING MODEL

This chapter describes in detail an application of the generic sizing model described in Chapter 4 to a data-centred business application developed in a fourth generation language. The chapter follows the framework of Chapter 4 with comments on particular issues of interest, or difficulties encountered, in applying the general principles and procedures of the model. Explanations of the statistical techniques used are included where appropriate. Chapter 6 gives three other less detailed examples of the application of the model to a system in the same application category developed using other software development technologies.

The application of the generic sizing model in a particular situation results in a *sizing model instance*. However, because this term is cumbersome, 'model' is frequently used throughout this chapter instead of 'sizing model instance'. The context should clarify any potential ambiguity resulting from this usage.

5.1 SOURCE DATA

The data used in this chapter is 'real world' data obtained from the development of a system of administrative applications at the New Zealand Correspondence School over the period 1986-1989. The earlier phases of this system have been described in detail in [VERN88]. Data used for the *development* of the sizing model, the reference data, was collected between 1986-87 while a later data sample, the test data, used for *validation* of the model was collected subsequently in late 1988 from later increments. In all something well in excess of 100,000 LOC have been developed for these applications. Though in many aspects the system is representative of its class of data-centred business applications, in some other respects it is not typical. In particular, both the data complexity and the processing complexity are high for this type of application. The Correspondence School caters for exceptions to the normal education system. These exceptions are many, varied, complicated and changeable, adding an extra complexity

dimension to the system. Though unrepresentative in these respects, the data from this system provides a stringent test of the model. One could say, if it will work for this data, it will work for any data-centred business application. A summary of the data used in this chapter, including both the reference data and the test data is shown in Table 5.1. The source data itself is listed in Appendix A (reference data) and Appendix B (test data).

Module Type	Reference data		Test data	
	n	LOC	n	LOC
Menus	12	252		
Screen Menus	25	1361	5	247
Screens	165	13070	41	2615
Relations	59	766	46	643
Reports	132	12659	41	5407
Updates	47	1455	29	797
Totals	440	29563	162	9709

Table 5.1 Application Data Used in this Case Study

Development data for real world systems is difficult to obtain because most organizations do not collect adequate or complete statistics of the development process or products. It is fortunate that the business application data used in this dissertation was available. Data from other application categories, such as real-time Ada systems, which are much less commonly developed in New Zealand, was unavailable. Business information systems and data processing applications make up the bulk of the software developed in New Zealand but there has been very little research into the size estimation of these types of systems.

Few of the sizing models in the published literature have been developed, or tested, using ideal source data sampled from a well-defined and stable population of applications. In general researchers have had to make do with whatever data can be collected, or is available. The data used in this study is no exception to this situation.

One particular, and inevitable problem, with source data in all sizing prediction situations is that models are developed using historical data for application to current developments. No software development technology is static. Inevitably some changes occur which affect the sizing prediction model, even if these are only minor. Developers acquire more experience, new techniques emerge, new releases of the system software and application

development software occur and users become more sophisticated in their demands. For these reasons, the population of systems, subsystems or increments to which a prediction model is applied is never quite the same, and may in some respects be rather different, from the reference population on which the prediction model was based. The user of any software sizing model must be aware of this potential problem and aware of changes in the environment that may necessitate changes to the model instance.

Some differences between reference and test populations occurred in this study. This resulted in one consequent change to the model instance developed here namely to the equation for the prediction of the size of relations. Differences between the reference data and test data populations are described as appropriate later in this chapter.

5.2 APPLICATION CATEGORY

A clear understanding of the essential nature of the category type is necessary before the precise characteristics of a development technology can be established. The following description of the application class to which the system of interest in this chapter belongs is closely based on [TATE88].

The application belongs to a relatively limited class of data processing systems which are nevertheless very common in business applications, and are important for this reason. They are concerned with **data** processing rather than data **processing**. This class may be characterized as

(i) data-centred
The main activities are keeping a database up to date and extracting information from that database as required. Module data dependencies are through the database or through a small number of parameters, rather than through a data flow network.

(ii) few major object classes
Most, if not all, of the modules which interact with the database can be regarded as instances of just a few object classes, such as menus, screens, file-to-file updates or reports. These objects inherit their shared behaviour from general, or in some cases particular, objects of each class which are further tailored to fit each specific instance.

(iii) simple state transition control mechanism

Inter-module control flow is typically through menus and can be represented as state transition diagrams.

(iv) comparatively low procedural complexity

This characteristic has several aspects. Firstly, application-specific manipulation, logic or calculation represents a relatively small proportion of the total job. However, since this part is specific and contains little that can be inherited or delegated, it may represent a rather larger proportion of the total effort. Secondly, the procedures embodying this logic are in general independent of each other and are called from, or embedded in, the database manipulation object instances. As noted in section 5.1, the application data used here is not entirely typical in this respect. Though structurally similar, the logic procedures tend to occur more frequently and to be more complex than usual.

Figure 5.1 gives a simplified diagram of the application class in a schematic data flow diagram form. The interaction process represents a menu or screen. The processes P1 to P5 represent embedded or called logic modules to meet application-specific needs. Some of these are only a handful of lines though in this case study some are quite large. Note that the main process types, A, B and C, are not connected directly by data flows, but all interact with the database. This is an oversimplification in that the menu-like control structure, which is not shown, effectively passes activating information from one process to the next; also, it is sometimes convenient to pass a few temporary variables from one process to the next. This is best shown on a modified state transition diagram. The system structures of interest are the data model and the state transition control diagram, not a data flow diagram.

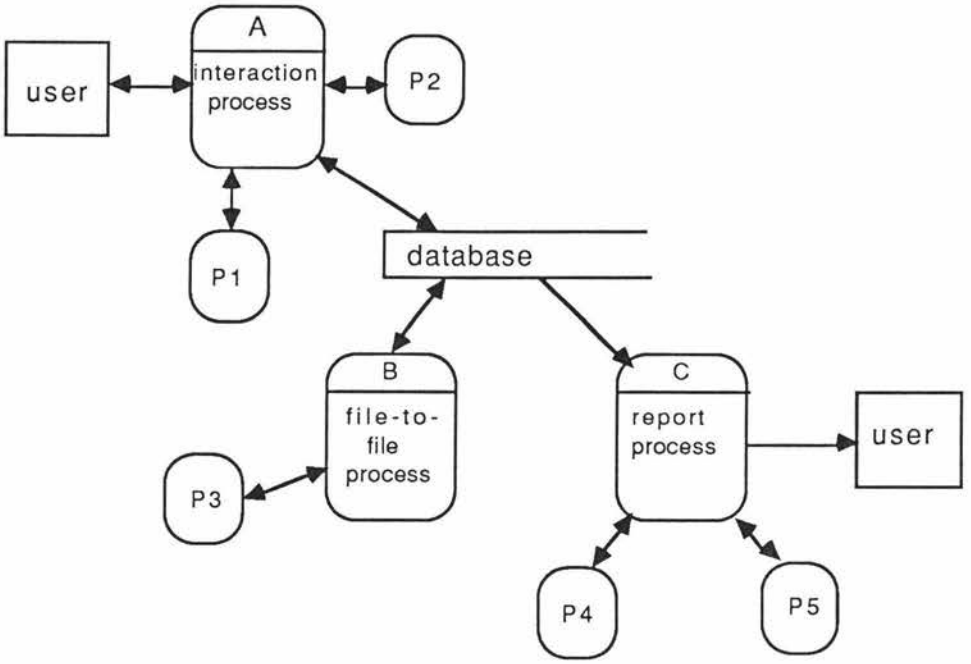


Figure 5.1 Schematic Data Flow Diagram of Application Class (from [TATE88])

This class includes many database applications, whether hosted from a language such as COBOL, SQL-based, or implemented using one of a number of 4GLs. Many business applications fall more or less into this class.

5.3 SOFTWARE DEVELOPMENT TECHNOLOGY

A software development technology which can be applied to an application category consists of the development methodology used together with the application development language and environment. However, from the point of view of the size estimation model many of the details of the development methodology are unimportant. The aspects of relevance are those which affect structure, or are size drivers. The development methodology also determines what sizing data is available and when this data is available. The division of a system into component types and components may be influenced both by the methodology and the language used.

The application, characterized as a data-centred business system, was defined using an entity-relationship data model (mapped into a relational data management system), together with associated dictionary definitions for the entities, relationships and attributes identified. Transactions and reports, interacting with the data base, were then defined for the various functions required of the system. The development strategy was that of incremental development. The data model was first defined as completely as possible then functionally related groups or subsystems of transactions, reports, etc. were developed in succession.

The software development language used for this system was the fourth generation language, ALL (Application Language Liberator) a Microdata product [MICR85] within a Pick environment. It is not the purpose of this dissertation to define the language ALL, aspects of which are only discussed where appropriate. Readers who wish to know more about the language ALL are referred in the first instance to [VERN88] where some simple annotated examples are given. This language uses a combination of procedural and non-procedural definitions to implement the system components. The non-procedural part is a 'fill-in-the-forms' 4GL system with the following module types: data screens, menu screens, updates, reports, file (relation) definitions and logic modules. The logic modules are written in a procedural language and have no independent existence, being embedded in the non-procedural modules that call them. The logic modules are programmed in a BASIC-like language.

5.4 PRIMITIVE SIZE MEASURE

Reference to Figure 4.2 shows that the choice of primitive size measure can be made subsequently to characterizing the software development technology. However, since it is convenient to discuss component partitioning, variable vectors and module size estimation equations together, the choice of primitive size measure precedes that discussion here.

LOC was chosen as the primitive size metric for the software development technology because of its *de facto* importance in this role. The LOC counts did not include comment lines and, because the model deals with component size estimation (and in this case components do not have associated explicit operating system command sequences), did

not include JCL. Each line was counted and no blank lines were included in the final LOC counts.

ALL is a mixed 4GL with both procedural and non-procedural parts. There is no established definition of a LOC counting convention for a fourth generation language of this type. Because of the lack of an existing LOC counting method for the non-procedural parts of these languages, which are not made up of lines in the conventional sense but of entries in predefined forms on a screen, a new line counting convention, which is discussed in more detail below, had to be invented. Logic lines in ALL are procedural and, because there is normally one statement per line, they can be counted directly.

5.4.1 Line Counting Convention for the Non-procedural Part

Table 5.2 shows the different module types used within the ALL development environment and the non-procedural form definitions required for each module type.

Line counting conventions for the non-procedural parts of the modules were investigated to obtain counting conventions that relate to the size of the modules. Similar form types are used by more than one module type so that similar counting conventions can be used across the different module types. To help choose the convention that best represents the actual size of the modules the number of tokens entered into the non-procedural modules was correlated with two different possible line counting conventions. This correlation was done for reports and screens, as the particular combination of forms was different for these two module types. In particular, report modules have definitions of the line types on the printed output but do not use definitions for screen layouts whereas screen modules have definitions of screen layouts but no line type definitions. The four definitions used in the remaining three module types are a subset of those used in screens. For this reason the other module types have not been investigated separately.

Module Type	Non-procedural Form Definitions Used					Procedural Code
screen	function	field	characterisitcs	screen		logic
report	function	field	characteristics		line types	logic
menu	function	menu		screen		
relation		file fields				
update	function		characteristics			logic

Table 5.2 Module Type Parts

The first counting convention counted physical lines, ie. counted every non-empty line across the page (or screen) as a single line and ignored whether entry of data was across or down the page, in columns or whether the entry tokens involved were related or not. Data obtained was stored in a variable called *LOC1*.

The second LOC counting convention counted logical lines defined as coherent small groups of one or more tokens referring to a single object such as file, data element, name, error procedure, date, etc. In some cases there were two or even three logical lines per physical line on a screen or on a page. This usually occurred when the information was arranged in columns. In other cases, for example, descriptions of some report lines containing long literals, one logical line could occupy several physical lines. The resulting LOC was called *LINES*.

The correlations between tokens and of both LOC1 and LINES are shown in Table 5.3. The relevant data is in Appendices A4 and A5.

	LOC1	*LINES*
Reports	0.94	0.85
Screens	0.91	0.88

Table 5.3 Correlations of Line Counts with Token Count

Because the counting convention for *LOC1* resulted in rather better correlations with token counts than that of *LINES* and seemed more natural, this counting convention was used as the basis for the non-procedural line counts.

Section 5.6 combines the consideration of module partitions, variable vectors and size estimation equations. However before this is done, the statistical techniques used in section 5.6 are briefly examined.

5.5 STATISTICAL TECHNIQUES USED

5.5.1 Development of Prediction Equations

The general method for obtaining component size estimations equations was multiple linear regression.

As a preliminary investigation into what variables might be included in these equations, a correlation of possible prediction variables with *LOC1* was used as a guide. However, some of the variables are applicable to only some of the component instances and hence although they may have only a low correlation with LOC1 nevertheless may be significant size drivers for some individual components. This is seen in the updates where control breaks only occur in a small percentage of components.

A number of regression equations were produced as candidates and investigated for each component type. In all several hundred of these equations were investigated but only the best of them are presented here. For each component type, the following statistical data is derived and presented.

(1) correlation for the variables of concern with *LOC1*
(2) one or more estimation equations
(3) an evaluation value set for each estimation equation (see 5.5.2)
(4) standard statistical significance tests[1] and related data

R^2	coefficient of multiple determination
F	variance ratio
s	residual standard deviation
n	number of cases
df	degrees of freedom

[1] The reader is referred to any standard statistical text for a definition of these terms.

t values, Student's t for each coefficient

In general, specific significance levels have not been given along with the F and t values. The F ratios are very large in all cases, with significance levels > .9999. The t values are also mostly large, with significance levels > .9995 though in a few cases there are t values of lower significance, which are usually accompanied by comments. In these circumstances it was felt most appropriate just to present the F and t values themselves.

The estimation equations presented in this chapter were chosen on a number of criteria:

(1) availability and suitability of data at the particular development phase
(2) ease of obtaining and counting data
(3) use of few estimating variables, consistent with estimation evaluation criteria
(4) intuitive correspondence to a general understanding of the data (naturalness)
(5) evaluation criteria discussed in 5.5.2.

Outliers were investigated but none were removed. Where appropriate, comments included with each component type discuss some of the outliers. This is a useful exercise leading to a better understanding both of the data itself and of the reasons why good size estimates are so difficult to obtain in some situations.

Because the evaluation of an estimation equation is a complex question it warrants separate discussion. 5.5.2 discusses this important question.

5.5.2 Prediction Model Evaluation Criteria

To evaluate a prediction model Conte et al [CONT86] suggest that the following five criteria, which are described briefly below, should each be considered, and the model giving the best results overall should be chosen. These criteria are briefly defined in sections 5.5.2(i) - (v).

coefficient of multiple determination, R^2,
mean relative error, RE*,
mean magnitude of relative error, MRE*,
prediction at the 25% level, PRED(.25),

root mean squared error, RMS and
relative root mean squared error, RMS*.

(i) Coefficient of Multiple Determination

R^2, the *coefficient of multiple determination*, indicates the extent to which a predicted value is related to an actual value. It is used to denote the percentage variance accounted for by the independent variables used in regression equations. The value of R^2 does not reflect how closely an actual value and an estimated value correspond to each other in an absolute sense. R^2 is used to indicate the extent to which an actual value and its predicted value are linearly related. R^2 is defined for n pairs of actual and estimated values as

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (\text{actual size}_i - \text{estimated size}_i)^2}{\sum_{i=1}^{n} (\text{actual size}_i - \text{mean actual size})^2}$$

(ii) Relative Error and Mean Relative Error

The *relative error* (RE) is investigated to see how well an actual value and its predicted value relate to each other. RE is defined as:

$$RE = \frac{\text{actual size - predicted size}}{\text{actual size}}$$

and the *mean relative error* (RE*) for n cases is:

$$RE^* = \frac{1}{n} \sum_{i=1}^{n} RE_i$$

small values of RE and RE* show that a model gives a good representation of size. However in practice large positive REs may be balanced by large negative REs hence a small RE* may not always be useful in practice. A large value of RE* can be a useful indicator of bias in a prediction model.

(iii) Magnitude and Mean Magnitude of Relative Error

Because of problems that can occur with RE and RE*, the *magnitude of the relative error* (MRE), and *mean magnitude of relative error* (MRE*), are also investigated. MRE is defined as

$$MRE = | RE | = \left| \frac{\text{actual size - predicted size}}{\text{actual size}} \right|$$

positive and negative errors will not cancel each other out and the smaller the value of MRE the better the prediction.

For a set of n cases MRE* is defined as

$$MRE^* = \frac{1}{n} \sum_{i=1}^{n} MRE_i$$

If MRE* is small the model produces on average a good set of predictions. Conte et al [CONT86] consider MRE* ≤ 0.25 as acceptable for effort prediction models. Similar criteria should apply to size prediction models.

(iv) Prediction at Level L - PRED(L)

PRED(L) is defined as

$$PRED(L) = \frac{k}{n} \text{ for k cases in a set n whose MRE} \leq L.$$

If PRED(.1) = 0.9 then 90% of the predictions for the cases in the set fall within 10% of their actual value. Conte et al [CONT86] suggest that an acceptable criterion for an effort prediction model is PRED(0.25) ≥ 0.75. There is no limit on the MRE of the estimates whose predictions exceed 25% of their actual values and it is possible for some very poorly predicted values to be present.

(v) Mean Squared Error and Relative Root Mean Squared Error

For a set of n cases *mean squared error* (SE*) is defined as

$$SE^* = \frac{1}{n} \sum_{i=1}^{n} (\text{Actual size}_i - \text{estimated size}_i)^2$$

This error represents the mean value of the error minimized by the regression model. From SE* the *root mean squared error* (RMS) can be calculated as

$$RMS = (SE^*)^{\frac{1}{2}} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\text{actual size}_i - \text{estimated size}_i)^2}$$

and the *relative root mean square error* (RMS*) is defined as

$$RMS^* = \frac{RMS}{\frac{1}{n} \sum_{i=1}^{n} \text{actual size}_i}$$

Conte et al suggest that RMS* $\leq$ 0.25 is an acceptable performance criterion for a prediction model.

Because there is no generally accepted standard for the evaluation of prediction models Conte et al (CONT86) suggest that the combination of MRE* $\leq$ .25 and PRED(.25) $\geq$ .75 can be used, these being the two most important of these criteria in most cases. In the tables presented with each component type in the following discussion values that fall within an acceptable level, using Conte's criteria, are shown in italics, while those that are the best for the particular group within which they fall are shown in bold type. Those values shown in bold and italic font are thus the best values for the group and also acceptable under the model evaluation criteria discussed above.

5.6 COMPONENT PARTITIONS

Two component partitions are investigated. The first partition is at the software requirements phase, and the second partition at the the preliminary design phase (see 3.4). The terms *Phase1* and *Phase2* are used below to refer to these two phases and their associated component partitions. Table 3.4 gives a brief list of the documents produced and components defined during the relevant phases of a typical lifecycle model. Although incremental development was used for the system under discussion, requirements were specified and the data base design was completed (for the reference data subsystems) before any software development started so that the software requirements phase is one that can be considered as appropriate to this particular development. The preliminary design, however, was done increment by increment. The reference data includes that of several early increments. The test data are sampled from a number of later increments.

5.6.1 Phase1

At Phase1 (in the sense of 5.6) the functions to be provided, user characteristics, development/operating/maintenance environments, programming language(s), tools and techniques have been defined. For this particular development technology the data base design, menu structures, screen and report layouts will be defined in the requirements phase. Because of the heavy dependence on menus, the menu structure is also defined early. The software development technology is naturally partitioned into the following component types:

menus
screens
relations
reports

As shown in Table 5.2, these component types have different definition characteristics within the ALL environment.

(i) Candidate Variable Vectors

Investigation of the above component types identified the following candidate variables for inclusion within the component vectors at Phase1.

Menus	*Screens*	*Reports*	*Relations*
choices	dictionary elements	report line types	data elements
	local variables	relations	
	data elements	dictionary elements	
	relations	local variables	
	choices	data elements	
		control breaks	
		sort fields	

In the above screen and report vectors, data elements are defined as the sum of dictionary elements and local variables. Candidate variables were investigated, within their component types, for their relationships with the size, in LOC, of components of that type. Multiple linear regression was the technique used for this investigation.

(ii) Menus

There were two different types of menus. The language provided a non-procedural module type called MENU which was very easy to program and required no procedural code in its development. This module type is referred to as *MMENU* for the rest of this discussion. Unfortunately *MMENU* modules proved to have performance problems. This resulted in the majority of menus being programmed using the ALL screen module type (called *SCRMENUS* for the rest of this discussion) instead of the menu module type. Two results are therefore shown for menus under the headings *MMENUS* and *SCRMENUS*.

Because the *SCRMENU* modules had quite different characteristics from the normal input and display screen modules (screens) menus programmed using the screen module machinery have been kept separate from the normal screen modules, which are discussed under (b) Screens below.

(a) SCRMENUS

The prediction equation and evaluation criteria for this component type, using the criteria suggested by [CONT86], are presented in table 5.3. The prediction equation gave very good results with most of the predicted sizes being very close to actual size, all the prediction criteria are well within the acceptable range, a high R^2, and F and t ratios which are at significance levels beyond the range of most statistical tables. PRED(.09) was 92%. Two cases, that could be considered outliers, with their MRE at .24 and .18, warranted further investigation. This revealed that one of these menus provided additional features and the other had extra security features. Data for the *SCRMENU* component type is included in Appendix A2.

SCRMENU Prediction Equation
LOC* = 23.6 + 3.99 choice

SCRMENU n	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
25	0.948	-0.009	0.052	1	4.37	0.080	416	13.4, 20.4	23	4.49

Table 5.4 SCRMENU Component Prediction Equation and Evaluation Criteria

(b) MMENUS

An excellent result was obtained with the prediction model for this component type which is presented in Table 5.5. The model predicted size within 6% of actual 100% of the time but PRED(.25) has been included in Table 5.5 for consistency.

MMENU Prediction Equation
LOC* = 7.77 + 2.04 choice

MMENU n	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
12	0.997	0.001	0.0122	1	0.386	0.018	3918	31.8, 62.6	10	0.42

Table 5.5 MMENU Component Prediction Equation and Evaluation Criteria

The original data for this component type is included in Appendix A2. Since *MMENUS* were not used later in the case study development, they are only included here for completeness and are not considered later.

The *SCRMENU* modules had much higher overheads in LOC than the *MMENU* modules and this is reflected in the constants of 24 and 7.8 respectively in their prediction equations. The system-supplied menu mechanism required very little programming. However, to do the same thing using the screen module machinery required a logic procedure to be called which had four lines of procedural code for each separate menu choice. This difference is reflected in the coefficients of the variable *choice*, 4 for *SCRMENUS* and 2 in *MMENUS* .

(iii) Relations

The results, which are shown in Table 5.6, were good. The variability within the relations is caused by definition of operator help messages for data elements (or attributes). If the particular attribute was used for the first time a message was normally defined while, if the attribute had been defined previously for some other relation, the message was already in the system dictionary and was automatically picked up. Relation data is included in Appendix A3.

RELATION Prediction Equation
LOC* = 1.37 + 1.88 data elements

RELATION n	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
59	0.978	-0.021	0.139	0.814	3.10	0.239	2496	2.78, 50.0	57	3.30

Table 5.6 Relation Component Prediction Equation and Evaluation Criteria

(iv) Screens

There were 165 screen modules available for analysis. The original data for this section is included in Appendix A4.

Screen object counts

The following object counts were identified as being possibly relevant to the size of screen modules:

dictionary elements
local variables
data elements = dictionary variables + local variables
relations
number of choices, if a small menu was included at the bottom of the screen

All of the above candidate variables, together with transformations of some of them, were correlated with the size in LOC of the screen modules. The correlations are shown in Table 5.7.

Candidate Variable	Correlation with LOC
data elements	0.73
√ data elements	0.71
data elements^2	0.74
relations	0.55
√ relations	0.53
relations^2	0.59
local variables	0.38
dictionary elements	0.64
choices	0.44

Table 5.7 Correlations of Screen Candidate Variables with LOC

There was not a great deal of difference between any of the correlations with the different versions of data elements or relations so it was decided to investigate several regression equations with combinations of data elements and (data elements)^2 and with relations and (relations)^2. Their prediction characteristics were investigated further through the five criteria suggested by [CONT86]. The results of the better combinations are shown in Table 5.8.

In this table the prediction equations are shown in groups with each member of the group having the same number of variables in its equation. The evaluation criteria are also shown in the same groups. The best value for each of the prediction model criteria, within a group, is shown in bold type and the best overall value for a particular criterion is shown in bold italic type. The prediction equation that gave the best results overall is

$$LOC^* = 26 + .134\ \textit{data elements}\ \hat{}2 + .618\ \textit{relations}\hat{}2 + 5.8\ \textit{choice}$$

which has a value of MRE* ≤ .25, a value of PRED(.25) just outside the the suggested 75% value at 70.3% and an RMS* just over the acceptable .25 value.

	SCREEN Prediction Equation
1	LOC* = 1.55 + 5.66 data elements
2	LOC* = 37.9 + 0 .189 data elements^2
3	LOC* = 4.15 + 4.51 data elements + 0.639 relations^2
4	LOC* = 26.2 + 0.157 data elements^2 + 4.75 relations
5	LOC* = 33.3 + 0.151 data elements^2 + 0.618 relations^2
6	LOC* = 26.0 + 0.134 data elements^2 + 0.618 relations^2 + 5.8 choice
7	LOC* = 1.54 + 3.96 data elements + 0.647 relations^2 + 5.17 choice

	R^2	RE*	MRE*	Pred(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	0.534	-0.087	**0.26**	**0.63**	29.6	0.37	186	0.25, 13.7	163	29.8
2	**0.547**	**-0.116**	0.29	0.59	**29.2**	**0.37**	197	10.2, 14	162	29.4
3	0.610	**-0.080**	**0.24**	**0.65**	27.1	0.34	126	0.733, 10.4, 5.62	162	27.3
4	0.597	-0.096	0.25	0.58	27.5	0.35	120	5.98, 10.8, 4.48	162	27.8
5	**0.620**	-0.100	0.26	0.58	**26.8**	**0.34**	131	9.4, 10.7, 5.46	162	27.1
6	***0.647***	-0.070	***0.22***	***0.70***	***26***	***0.3***	98.2	6.58, 9.24, 5.67, 3.67	161	26.1
7	0.632	***-0.057***	0.23	0.62	26.3	0.33	92.2	.276, 8.70, 5.84, 3.14	161	26.6

Table 5.8 Screen Component Prediction Equations and Evaluation Criteria

Investigation of outliers revealed that the number of logic lines has a major effect on the size of the code for a screen. This logical complexity cannot be estimated at Phase1 before the relevant design issues are considered. Complexity can be subjectively estimated on an ordinal scale depending on how much is known about the screen. Three or five level scales can conveniently be used. This idea is developed further in 5.6.2 where Phase2 predictions are considered.

(v) Reports

132 report modules were available for investigation. Data for this component type is included in Appendix A5. The following candidate variables were investigated for possible inclusion in the prediction equation:

Candidate Variable	Correlation with LOC
line types	0.916
line types^2	0.900
relations	0.402
relations^2	0.371
data elements	0.895
data elements^2	0.940
sort fields	0.025
control breaks	0.279

Table 5.9 Correlations of Report Candidate Variables with LOC

Line types, (data elements)^2 and *relations* were identified as the most promising variables for initial investigation. The equations in Table 5.10 were chosen, after investigation, as giving the best results with an increasing number of variables shown in the prediction equations. Evaluation criteria for each of these prediction equations are also shown in Table 5.10.

	REPORTS Prediction Equations
1	LOC* = 41.3 + .124 data elements^2
2	LOC* = 35.5 + .0794 data elements^2 + 2.52 line types
3	LOC* = 7.69 + .051 data elements^2 + 3.57 line types + 6.85 relations
4	LOC* = 12.1 + .047 data elements^2 + 3.61 line types + 8.92 relations - 4.84 sort fields

	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	0.88	-0.18	1.82	0.52	32.9	0.34	988	12.5, 31.4	130	32.4
2	0.91	-0.17	0.33	0.49	28.08	0.3	664	11.7, 10.2, 6.36	129	28.4
3	**0.94**	**-0.08**	***0.20***	**0.73**	23.31	*0.25*	657	1.73, 6.76, 10.1, 7.6	128	23.7
4	**0.94**	**0.08**	*0.23*	0.65	**22.53**	***0.24***	525	2.65, 6.46, 10.5, 8, 2.99	127	23.0

Table 5.10 Report Component Prediction Equations and Evaluation Criteria

The values shown in bold are those that give the best results for that particular criterion while those shown in italics are those considered acceptable under the model evaluation criteria. Using the model evaluation criteria, equation (3) from Table 5.10 is the best equation overall although there is not a great deal of difference between equations (3) and (4).

Because there were two quite different types of reports, statistical and non-statistical, they were investigated separately to see if instead of using just one vector of estimation variables for reports, with a single prediction equation, there should be two component types, statistical reports and non-statistical reports each with its own prediction equation.

The statistical reports were all rather larger than the non-statistical reports; the mean for the statistical reports being 357 LOC while the mean for the non-statistical reports was 71; the overall mean for all reports was 96. The reports were divided into the two groups and individual regression equations were developed for the two different types of reports.

The negative coefficient for sort fields in equation (4), significant at the .995 level, requires comment. Reports with several sort fields tended to be more straightforward standard control-break reports whereas some of those with few sort fields tended to have a more complex and less regular structure and logic.

(a) Non-statistical Reports

There were 120 non-statistical reports available for analysis. Correlation of candidate estimation variables with LOC are shown in Table 5.11. The prediction equations shown in Table 5.12 were those that gave the best results. By comparison with Tables 5.9 and 5.10 for all reports, it will be seen that, while the individual variable correlations with LOC in 5.11 are not as good as those of 5.9, the evaluation criteria of 5.12 are better than those of 5.10, though perhaps not sufficiently better to warrant splitting the report component types into two.

Candidate Variable	Correlation with LOC
line types	0.502
relations	0.597
relations^2	0.612
data elements	0.767
data elements^2	0.727
sort fields	0.390
control breaks	0.314

Table 5.11 Correlation of Candidate Variables with LOC for Non-statistical Reports

Equation (1) from Table 5.12 overall gave the best predictions. The prediction equation of (1) is somewhat different from the best prediction equation for all reports with data elements, rather than (data elements)^2, included as one of the independent variables. The other independent variables have coefficients that are very similar to those in the prediction model for all reports. RMS* is significantly reduced when non-statistical reports are investigated on their own.

Similar comments to those in (v) above for all reports are relevant with regard to the negative coefficients for sort fields in Table 5.12. The t-ratios for sort fields in equations (1) and (2) are low, however, with significance levels of only .95 and .9 respectively, suggesting that the sort fields effect is only just noticeable in these cases.

	NON-STATISTICAL REPORTS Prediction Equations
1	LOC* = 12.5 + 1.35 data elements + 3.76 line types + .807 relations^2 - 2.10 sort fields
2	LOC* = 20.4 + .032 data elements^2 + 3.94 line types + .828 relations^2 - 1.73 sort fields
3	LOC* = 7.46 + .042 data elements^2 + 3.59 line types + 8.65 relations - 2.90 sort fields
4	LOC* = -1.96 + 1.71 data elements + 3.4 line types + 8.35 relations - 3.22 sort fields

	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	**0.81**	-0.06	***0.193***	***0.75***	**16.74**	***0.24***	124	3.37, 4.83, 10.2, 9.42, 1.78	115	17.10
2	**0.81**	-0.06	*0.209*	0.72	17.18	*0.25*	122	5.87, 4.6, 11.1, 9.75, 1.44	115	17.24
3	0.80	-0.06	*0.217*	0.70	17.38	*0.25*	117	1.89, 6.45, 10.4, 9.35, 2.23	115	17.56
4	0.80	**-0.05**	*0.217*	0.67	17.5	*0.25*	115	0.5, 6.29, 9.32, 8.71, 2.46	115	17.67

Table 5.12 Non-statistical Report Component Prediction Equations and Evaluation Criteria

(b) Statistical Reports

Twelve statistical reports, making up a fairly homogeneous group, were available for analysis. The correlations in Table 5.13 were found between the various candidate variables and LOC for this small group of reports.

Candidate Variable	Correlation with LOC
line types	0.974
relations	0.792
relations^2	0.798
data elements	0.810
data elements^2	0.916
sort fields	0.804
control breaks	0.398

Table 5.13 Correlation of LOC with Candidate variables for Statistical Reports

The following prediction equation was identified as the most useful predictor of LOC.

STATISTICAL REPORTS Prediction Equation
134.6 + 9.88 relations + 3.51 line types

n	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
12	0.98	-0.002	0.026	1	12.14	0.034	240	12, 4.07, 13.2	9	14.02

Table 5.14 Evaluation Criteria for Prediction of Statistical Reports

When compared to the model evaluation criteria, all of the values fall well within acceptable bounds for a prediction model. PRED(.25) is at 100%, MRE* at .026 and there is a very low RMS* of .034. PRED(.11) is 100%. When either data elements or (data elements)^2 are included in the prediction equation they give negative coefficients. This seems rather unnatural. In view of the small amount of fairly homogeneous data available it may not be possible to obtain a reliable prediction equation from such a small sample even though the evaluation criteria appear to be excellent.

5.6.2 Later Predictions at Development Phase2

Phase2 is early in preliminary design, if the life cycle model of Table 3.4 is adopted. In the incremental development mode used, different subsystems were designed at different times.

During this phase more detailed knowledge is available. The component partitions identified are those of the earlier partition with the addition of **updates** (which implement file-to-file processes). Since updates do not involve outputs to, or inputs from, the user, they are not visible at the earlier specification stage (Phase1). There is no need to develop additional prediction equations for menus and relations as those already developed show good enough results when their evaluation criteria are investigated. Moreover, no further information is available concerning them.

More detailed knowledge (functional descriptions) of reports and screens are available at Phase2, although this includes few more objectively countable variables. However, the component prediction equations are likely to be improved if the additional knowledge can be used in a subjective fashion to give some estimate of the complexity of the logic required by a specific component in comparison with other components of the same type. As noted in section 5.1, the complexity of logic in this case study is unusually high for its application category

Complexity Measures

Two complexity measures, a three level (*cplx3*) and a five level (*cplx5*) complexity measure, were defined for each of the three component types, Screens, Reports and Updates, investigated further in Phase2. Values assigned for these complexity measures were derived from the number of lines of procedural code in the logics of the various components within each component class. In each case the numbers of logic lines of each component were ranked and then divided into three or five groups of equal size. Each member of a group was then assigned the value of the median member of the group.

(i) Screens

In addition to the variables identified earlier there are three others that may usefully be included in prediction equations for screens. The first is the number of logical screens

that make up a single screen and the other two are the complexity variables cplx3 and cplx5. Logical screen are 'screens within screens', ie. portions of a screen with different format patterns, often for line items or other repeating groups, or for separate screen functions such as a small bottom-of-screen menu. Values given for cplx3 and cplx5 are shown in the following tables.

Screen cplx3		
level	range	median
1	0-11	0
2	12 - 29	17
3	30+	38

Screen cplx5		
level	range	median
1	0-5	0
2	6 - 16	11
3	17 - 30	23
4	31 - 61	41
5	62+	95

Tables 5.15 and 5.16 summarize the relevant correlations, prediction equations and evaluation criteria using the same conventions as previously. The logical screens have low correlation and, when tried in prediction equations, are not significant. Cplx5 gives rather better estimates than cplx3 though this may be offset somewhat in practice by the slightly greater degree of difficulty in reliably assigning a 5-level complexity as against a 3-level complexity, in a situation of incomplete evidence. The best prediction equation with cplx3 is equation (3), and that with cplx5 is equation (5).

Candidate Variable	Correlation with LOC
logical screens	0.22
cplx3	0.78
cplx5	0.85

Table 5.15 Correlation of Phase2 Screen Candidate Variables with LOC

	SCREEN Prediction Equation
1	LOC* = 19.3 + 0.108 data elements^2 +3.51 relations + 1.15 cplx3
2	LOC* = 6.21 + 2.89 data elements + 0.539 relations^2 + 1.13 cplx3
3	LOC* = 24.2 + 0.101 data elements^2 + 0.508 relations^2 +1.14 cplx3
4	LOC* = 27.0 + 0.0797 data elements^2 + 0.400 relations^2 + 0.900 cplx5
5	LOC* = 22.2 + 0.0816 data elements^2 + 3.11 relations + 0.918 cplx5
6	LOC* = 29.4 + 0.0983 data elements^2 + 0.958 cplx5
7	LOC* = 23 + 0.099 data elements^2 + 0.513 relations^2 + 1.29 choice + 1.09 cplx3

	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	0.72	**0.023**	*0.16*	***0.82***	22.7	0.29	136	5.10, 7.93 3.88, 8.25	161	23.4
2	0.72	0.080	*0.24*	0.66	26.9	0.34	140	1.3, 6.94, 5.57, 8.14	161	23.1
3	**0.74**	-0.035	***0.12***	***0.82***	**22.2**	***0.2***	150	7.73, 7.65, 5.35, 8.55	161	22.5
4	**0.86**	-0.025	*0.12*	*0.90*	**16.4**	***0.21***	321	12.2, 8.19, 5.66, 16.4	161	16.6
5	0.85	**0.020**	***0.11***	***0.91***	16.8	***0.21***	302	8.20, 8.12, 4.72, 16.4	161	17.1
6	0.83	0.030	*0.13*	*0.87*	18.0	*0.23*	390	12.5, 9.86, 16.3	162	18.1
7	0.74	0.032	*0.15*	0.74	22.1	0.28	113	6.69, 7.45, 5.38, .862, 7.46	160	22.5

Table 5.16 Phase2 Prediction Equations and Evaluation Criteria for Screens

An examination of the Phase2 screen outliers shows two main types. First, a set of very simple maintenance screens generated automatically from relation definitions by the system BUILDER process. These, even following minor modification, have much lower than normal screen complexity, both in logic and in numbers of logical screen specifications. The other set are abnormally complex screens, often with five or six pages of logic, which is out of all proportion to the data elements and relations they process. These outliers reflect the wide range of purposes for which screens are used, and illustrate the difficulty of prediction with a very diverse population.

(ii) Reports

Additional variables identified at Phase2 for possible inclusion in prediction equations are the number of logical reports and the two complexity variables. The values assigned for cplx3 and cplx5 were as follows:

Report cplx3		
level	range	median
1	0-11	5
2	12 - 43	20
3	44+	70

Report cplx5		
level	range	median
1	0 - 6	3
2	7 - 15	7
3	16 - 24	20
4	25 - 63	46
5	63+	82

The prediction equations for all reports and their evaluation are shown in Table 5.17. Logical reports (a measure of embedded one-to-many or other structures within the report data) are not significant. There is very little difference between cplx3 and cplx5 so, for simplicity and ease of use, the former is preferred. As can be seen from the small corresponding t-ratios, significant at only .9 and .95 respectively, the constant terms in these two estimation equations are barely significantly different from zero.

REPORTS	Prediction Equations
1	LOC* = 5.47 + .040 data elements^2 + 3.4 line types + 3.09 relations + .82 cplx3
2	LOC* = 6.22 + .037 data elements^2 + 3.38 line types + 2.89 relations + .82 cplx5

	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	**0.97**	**0.002**	***0.13***	***0.91***	16.73	***0.17***	978	1.39, 7.37, 13.3, 4.21, 10.9	127	17.06
2	**0.97**	-0.003	***0.13***	*0.88*	**16.68**	***0.17***	985	1.94, 6.67, 13.3, 3.91, 11	127	17.00

Table 5.17 Phase2 Prediction Equations and Evaluation Criteria for Reports

(a) Non-statistical Reports

Investigation of non-statistical reports results in the prediction equations shown in Table 5.18 giving the best results. The values assigned for cplx3 and cplx5 are shown below.

Non-statistical cplx3		
level	range	median
1	0 - 7	4
2	5 - 25	17
3	26+	61

Non-statistical cplx5		
level	range	median
1	0 - 5	3
2	6 - 13	7
3	14 - 21	17
4	22 - 52	28
5	53+	68

In this case there is again little difference between equations including cplx3 and cplx5, with the former giving slightly better results

	NON-STATISTICAL REPORTS Prediction Equations
1	LOC* = 3.62 + 1.52 data elements + 1.83 line types + 2.48 relations + .87 cplx3
2	LOC* = 6.17 + 1.11 data elements + 2.37 line types + 2.61 relations + .996 cplx5

	R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
1	**0.94**	**-0.010**	*0.106*	***0.95***	**9.65**	***0.14***	433	1.65, 10.0, 8.42, 4.93, 16.6	115	9.85
2	0.93	-0.017	***0.096***	*0.94*	10.11	***0.14***	391	2.64, 6.80, 10.9, 4.95, 6.80	115	18.30

Table 5.18 Prediction Equations and Evaluation Criteria for Non-statistical Reports

An investigation of the report outliers showed substantial logical complexity variations largely independent of the other variables, data elements, line types and relations and often related to the handling of complicated exceptions. Similar comments to those for screens are appropriate.

(b) Statistical Reports

The original group of 12 statistical reports was a fairly homogeneous group. This meant that it was difficult to assign complexity ratings in a similar manner to that done earlier for the other component types. A tentative scale, as shown following, was set up but it

is based on too small a sample and is likely to need modification as additional data becomes available.

Statistical Reports Cplx3		
level	range	median
1	0-100	86
2	101-250	200
3	251+	291

Neither the number of logical reports nor the cplx3 variable were found to be significant when they were brought into the equations. The Phase2 prediction equations for the statistical reports are therefore the same as the Phase1 equations.

(iii) Updates

This component type is included only in the later of the two partitions, ie. at Phase2. These modules perform file-to-file updates and are unlikely to have been defined to the necessary level of detail for any component size estimates until this stage. In many cases their very existence will not have been noticed until Phase2. Updates tend to be used for various file cross-referencing or reformatting operations. They have relatively few non-procedural lines, mainly specifying the files involved. The size of the update components is thus very dependent on the size of their associated logic. 47 cases were available for analysis. The source data is listed in Appendix A6. The correlations with LOC shown in Table 5.19 were obtained and the three and five level complexity values were as follows:

Updates cplx3		
level	range	median
1	0 - 3	0
2	4 - 6	5
3	7+	29

Update cplx5		
level	range	median
1	0	0
2	1 - 3	3
3	4 - 6	5
4	7 - 40	10
5	40+	80

Candidate Variable	Correlation with LOC
logical updates	-0.107
relations	0.332
relations^2	0.393
data elements	0.889
data elements^2	0.909
sort fields	-0.078
control breaks	0.052
cplx3	0.926
cplx5	0.980

Table 5.19 Correlations of Update Candidate Variables with LOC

UPDATES Prediction Equation
LOC* = 11.05 + 4.82 control breaks + 1.02 cplx5 + 3.01 logical updates

R^2	RE*	MRE*	PRED(.25)	RMS	RMS*	F-ratio	t-ratios	df	s
0.97	-0.02	***0.132***	***0.91***	**4.69**	***0.15***	401	4.37, 1.86, 34.5, 1.55	43	4.9

Table 5.20 Prediction Equations and Evaluation Criteria for Updates

The correlations of update variables with LOC are shown in Table 5.19. There are, however, high correlations between data elements and cplx3 (.93) and between data elements and cplx5 (.90). This accounts for the fact that a data elements term does not occur in the prediction equation of Table 5.20. The dominant term is cplx5, the effect of control breaks and logical updates being much less significant as shown by their low t-ratios (.95 and .9 significance levels respectively).

5.6.3 Prediction Equation Summary

Tables 5.21 and 5.22 shown below summarize the prediction equations obtained for each of the component partitions (Phase1 and Phase2).

PHASE 1 ESTIMATION EQUATIONS	
Component	Equation
Menu	LOC* = 23.6 + 3.99 choice
Relation	LOC* = 1.37 + 1.88 data elements
Screen	LOC* = 26 + 0.134 data elements^2 + 0.618 relations^2 + 5.8 choice
Report	LOC* = 7.69 + 0.051 data elements^2 + 6.85 relations+ 3.57 line types
Non-statistical Rept	LOC* = 12.5 + 1.35 data elements + 0.807 relations^2 +3.76 line types-2.10 sort fields
Statistical Report	LOC* = 134.6 + 9.88 relations + 3.51 line types

Table 5.21 Phase 1 Estimation equations Summary

PHASE 2 ESTIMATION EQUATIONS	
Component	Equation
Menu	LOC* = 23.6 + 3.99 choice
Relation	LOC* = 1.37 + 1.88 data elements
Screen	LOC* = 22.17 + 0.0816 data elements^2 + 3.11 relations + 0.918 cplx5
Report	LOC* = 5.47 +0.040 data elements^2 + 3.09 relations+ 3.4 line types + 0.82 cplx3
Non-statistical Repc	LOC* = 3.62 + 1.52 data elements + 2.48 relations + 1.83 line types + 0.87 cplx3
Statistical Report	LOC* = 134.6 + 9.88 relations + 3.51 line types
Update	LOC* = 11.05 + 4.82 control breaks + 3.01 logical updates + 1.02 cplx5

Table 5.22 Phase 2 Estimation Equations Summary

As can be seen from the very large F values, and the large t values, in the evaluation criteria tables for the Phase1 and Phase2 model instances, the regression is accounting for almost all the variability and the sum of squares of the residuals is comparatively small. The statistical significance levels associated with the related hypotheses that such relationships exist and that the coefficients in them are different from zero are thus very

high, in most cases beyond the levels of available tables, ie. >> .995 for the F values, and mostly >.995 for the t-ratios. This merely confirms the indications given by the other evaluation criteria that the model instances are a very good fit to the reference data.

It is debatable in this case whether there is sufficient difference between the Phase2 and Phase1 models to warrant using the former, apart from the addition of updates. In practice, however, the Phase2 exercise is advisable because it necessitates a new and more careful examination of the data which might well have changed and which should be more complete than the Phase1 data. A discussion of the additive form of the regression equations and the use of squared terms within these equations is included in Appendix D.

5.7 APPLICATION OF THE DERIVED MODEL INSTANCE TO SIZE PREDICTION OF TEST COMPONENTS

A sample of data from later increments of the same system was taken in order to test the sizing model instance developed in 5.1 to 5.6 using the generic model of Chapter 4. Basic details of the sample are given in Table 5.1. Except in the case of screen menus (SCRMENUS), where only five were available, the sample, though small, was representative, being randomly selected. The sample included two in five relations, one in two screens, one in ten reports and one in two updates. In sampling theory terms, this is essentially a stratified sample [DEMI60].

As noted in 5.1, data from later increments is not from the same population as data obtained during the initial development of a system. It must be emphasized again that the user of any model instance must be sensitive to the possibility of such differences. See in particular the change in relation sizes in 5.7.1 (ii) below.

In each of the result tables shown below the criteria suggested in [CONT86] have been calculated for the predictions with the new set of test data. Results for the prediction equations with the original or reference data have also been included in the result tables for comparative purposes.

5.7.1 Phase1 Prediction

(i) Prediction of Menu Sizes

As discussed earlier there is only one prediction equation for menus. This equation was applied to the five new menus to predict their size. This small set included all menus to be developed since the 1986-7 data was collected. The excellent results from this prediction are shown in Table 5.23.

MENUS	n	RE*	MRE*	Pred(.25)	RMS	RMS*	mean
Test data	5	0.003	0.072	1	4.100	0.083	49.4
Reference data	25	-0.050	0.160	1	4.370	0.080	54.4

Table 5.23 Prediction Results for Menus (Phases 1 and 2)

LOC* = 247 which is 100% of the actual LOC of 247.

(ii) Prediction of Relation Sizes

The relations shown in the table below were chosen by taking a random two in five sample from a large number of new and revised relations developed in 1988. During this time a system error was introduced in a revision of the development software. This error mixed up help messages in some cases for the data elements in the relations. As a result, the user had not only not added help messages to these new relations but had deleted the help messages from the original relations analysed earlier in 5.6.1 (ii). A consequence of this change was that the original and now outdated prediction equation gave the following much overestimated results (note the large RE* and an MRE* of similar magnitude) shown in Table 5.24.

RELATIONS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test Data	46	-0.692	0.692	0.00	29.40	2.100	13.98
Reference Data	59	-0.021	0.139	0.81	3.10	0.239	12.98

Table 5.24 Prediction Results for Relations Using Original Equation

However if a new prediction equation is developed for the original data with the help messages excluded then a new equation

$$LOC^* = 1.18 + 1.00 \text{ data elements}$$

is obtained. Results for the new prediction equation with the now changed original reference data and the test data set are shown below in Table 5.25. In spite of the fact that, on average, most of the test data relations have six more data elements than the reference data relations, as indicated by their mean sizes, the prediction results are excellent. Pred(.25) is 1 and LOC* = 636 which is 99% of the actual LOC of 643.

RELATIONS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test Data	46	0.010	0.061	1	0.48	0.030	13.98
Reference Data	59	-0.010	0.062	1	0.38	0.050	7.49

Table 5.25 Prediction Results for Relations Using Modified Equation (Phases 1 and 2)

(iii) Prediction of Screen Sizes

Using what would appear to be the best prediction equation identified from those presented in 5.6.1(iv) namely

$$LOC^* = 26 + 0.134\ \textit{data elements}\text{^}2 + 0.618\ \textit{relations}\text{^}2 + 5.8\ \textit{choice}$$

the results shown in Table 5.26 were obtained.

SCREENS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test Data	41	-0.071	0.227	0.68	25.72	0.403	63.76
Reference Data	162	-0.070	0.220	0.70	26.00	0.300	80.68

Table 5.26 Phase1 Prediction of Screens

The test data screens are rather smaller than the reference data screens, 17 lines on average, or 21% smaller. This is because most of the large data entry screens were produced during the early stages of development and there is a much smaller proportion of such screens in later increments. Nevertheless, the test data predictions have evaluation characteristics similar to those of the reference data. Although Pred(.25) is only .68, LOC* = 2593 which is 99% of the actual of 2615.

(iv) Prediction of Report Sizes

(a) All Reports

Tables 5.27 - 5.29 show test data prediction results for all reports, non-statistical reports and statistical reports. As can be seen from an examination of the mean sizes, the non-statistical reports in the test data are 43% larger (30 LOC) on average than those in the reference data. However, the statistical reports in the test data are smaller than the reference data, but probably not significantly so, given the small number of cases involved. For all reports, there is a 32% increase in average size. This trend is probably typical of the increasing sophistication of reports in later increments of an on-going application development.

In view of the test and reference population differences, it is not surprising that the evaluation criteria of Tables 5.27 - 5.29 are worse than those for the reference data, though they are not dramatically so.

ALL REPORTS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	41	-0.06	0.250	0.61	53.25	0.40	132
Reference data	132	-0.08	0.200	0.73	23.31	0.25	96

Table 5.27 Phase1 Prediction of Reports

Use of the Phase1 prediction equation to estimate all reports results in a LOC* = 5126 which is 95% of the actual of 5407.

(b) Non-statistical Reports

NON-STATISTICAL REPORTS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	34	0.01	0.270	0.62	58.33	0.58	100
Reference data	120	-0.06	0.190	0.75	16.74	0.24	70

Table 5.28 Phase1 Prediction of Non-statistical Reports

As shown in Table 5.28, use of the Phase1 prediction equation to estimate the size of the non-statistical reports results in a LOC*= 2829 which is 83% of the actual 3397.

The relatively poor prediction in this case is caused by changes in the population between the reference and the test data. There is, however a lesson to be learnt from a comparison of the predictions of all reports and those for non-statistical reports. A component type which is very narrow and specialized may give better estimates for a stable population but it may also be more sensitive to changes in the population of components of that type.

(c) Statistical Reports

STATISTICAL REPORTS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	7	0.022	0.124	0.86	46.51	0.162	287
Reference data	12	-0.002	0.026	1	12.14	0.034	357

Table 5.29 Phase1 Prediction of Statistical Reports

Use of the Phase1 prediction equation for the estimation of statistical reports results in a LOC* = 1927 which is 96% of the actual 2010.

Table 5.33 summarizes the effect of partitioning reports into two separate component types and of treating them as a single component type. This table shows that the Phase1 estimates of reports which are treated as a single component type gives a better result (95% of actual) than if reports are separated into two component types (88% of actual) and the two component type estimates summed. The point made in (b) above is also relevant here.

5.7.2 Phase2 Prediction

Tables 5.30 - 5.34 show comparisons between the Phase2 predictions of the test data and the reference data. Similar comments to those of 5.7.1 apply, though the results are slightly better for reports and slightly worse for screens. This is a little surprising as one would expect a later phase with increased knowledge of components would result in substantially better estimates for all components. However, the effects of population changes are, by their nature, unpredictable.

(i) Prediction of Screen Sizes

SCREENS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test Data	41	-0.024	0.133	0.85	17.41	0.278	63.76
Reference Data	162	0.020	0.110	0.91	16.80	0.210	80.68

Table 5.30 Phase2 Prediction of Screens

Pred(.25) has improved from the .68 of Phase1 to .85, however LOC* is now 2570 which is a drop from the earlier 99% of actual to 98%.

(ii) Prediction of Report Sizes

(a) All Reports

REPORTS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	41	-0.02	0.190	0.71	46.92	0.36	132
Reference data	132	-0.01	0.106	0.95	9.65	0.36	96

Table 5.31 Phase2 Prediction of Reports

Estimation of total size for all reports shows a slight improvement from the earlier 95% of actual to 96%. Pred(.25) has also increased from the earlier .61 to .71.

(b) Non-statistical Reports

The Phase1 Pred(.25) of .62 has now improved to .85 and LOC* has also improved from the earlier 83% to 89%.

NON-STATISTICAL REPORTS	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	34	0.04	0.170	0.85	48.28	0.48	100
Reference data	120	-0.01	0.106	0.95	9.65	0.36	70

Table 5.32 Phase2 Prediction of Non-statistical Reports

Table 5.33 shows the effect on estimation, by phase, of the separation of reports into two component types. Note that the estimation of all reports as a single component type in Phase2 gives better results (96% of actual) than does the splitting of reports into two component types (91% if actual). This is also true of the Phase1 estimates. The population changes between the reference and test data have a greater effect on the more specialized size estimates.

	Actual	Phase1 prediction	%	Phase2 prediction	%
non-statistical	3397	2829	83	3009	89
statistical	2010	1927	96	1927	96
Total	5407	4756	88	4936	91
All reports	5407	5126	95	5212	96

Table 5.33 Summary of LOC* for Report Component Types

(iii) Prediction of Update Sizes

There was some change in nature of the updates in the test data from that in the reference data. Updates had initially been mostly used rebuild file indexes as a result of changes

made through screens. The test data updates often have quite a different function in that many of them are used for changing data into different file formats required by outside agencies. The index rebuilding is now mainly done through the screens, on-line. This population change in the updates results in an estimate with Pred(.25) of only .64 and the poorest LOC* of any component type. The estimate of 702 LOC is only 88% of actual.

UPDATES	n	RE*	MRE*	PRED(.25)	RMS	RMS*	mean
Test data	28	0.04	0.209	0.64	10.58	0.385	27
Reference data	47	-0.05	0.125	0.85	6.13	0.198	31

Table 5.34 Prediction of Updates

5.7.3 Summary of Prediction

Table 5.35 gives an overall summary of both the Phase1 and Phase2 predictions for the test data. In spite of the test population differences noted in 5.7.1 and 5.7.2, the overall estimates at both phases are excellent. These results suggest considerable robustness of the model in the face of quite large changes in some of the component type populations.

PHASE 1 ESTIMATES	n	LOC	Estimate	Pred%
Menus	5	247	247	1.00
Relations	46	643	636	0.99
Screens	41	2615	2593	0.99
Reports	41	5407	5126	0.95
Total LOC		8912	8602	0.97

PHASE 2 ESTIMATES	n	LOC	Estimate	Pred%
Menus	5	247	247	1.00
Relations	46	643	636	0.99
Screens	41	2615	2570	0.98
Reports	41	5407	5212	0.96
Updates	29	797	702	0.88
Total LOC		9709	9367	0.96

Table 5.35 Summary of Estimates for Phases 1 and 2

5.8 CALIBRATION OF A NEW FPA-LIKE METRIC

A new function point metric, derived using the model instance in this chapter, will be called the VFP, to distinguish it from the Albrecht FPA metric, for which the traditional abbreviation FP will be used. The software development technology described in this chapter will be called technology B; the reference technology, characterized by the FPA component type partition, variable vectors and weights, will be called technology A. Lines of code in technology B will be denoted by BLOC and estimates of them by BLOC*. The calibration method follows section 4.4. However one practical matter must be dealt with first, namely ensuring that the partial sizes measured or estimated by the two technologies are the same.

5.8.1 Matching Partial Sizes

In technology B one is concerned with the following partial sizes:

(i) the Phase1 partial size consisting of the total BLOC for all relations, menus, screens and reports (28100 BLOC for the reference data of Table 5.1)

(ii) the Phase2 partial size, in which the additional component type, updates, is added to the Phase1 total (29563 BLOC)

(iii) the total system size of interest which is approximately 10% higher than the Phase2 size. This is made up of account security, port and access management, periodic data base reorganization, JCL command sequences for overnight and other batch jobs, and a miscellaneous set of systems programming jobs. These are not all documented or collected in one place, nor can they all be completely identified in the particular environment. The approximate total reference system size is 32519 BLOC.

(iv) The FP count of the reference data set measured using the standard FPA reference technology A, which can be imposed on the technology B component partition. The relevant correspondences are shown below

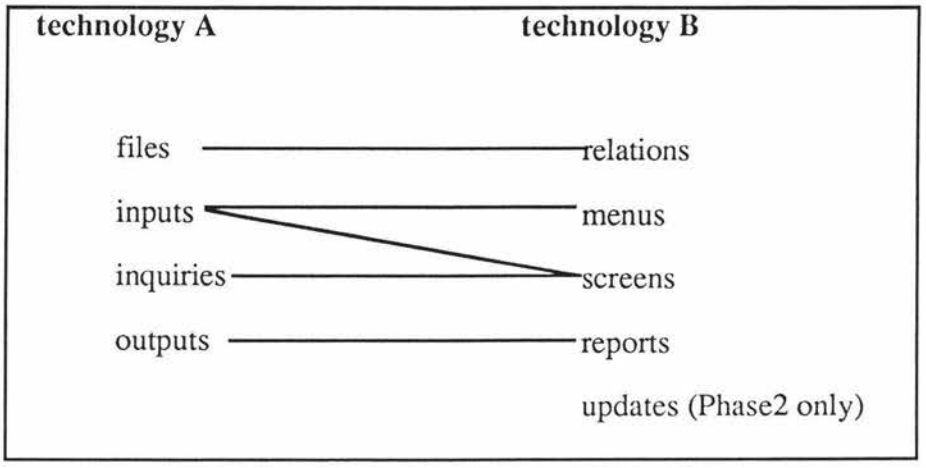


Figure 5.2 Correspondences Between Technologies A and B

Some screens are inputs and some inquiries. Updates are not represented at all in technology A. The FP count, using FPA, is 2575. Since the adjustment factor was 1.0, and therefore no adjustment up or down of the raw FP count was necessary, this represents an estimate of the total size in FPs. One disturbing feature of the FP count for this case was the proportion of screens and reports in the high category. FPA has three fixed categories low, average and high (see 2.5.1). The criteria for these, set out in Table 2.1, were applied rigidly. Nevertheless, the great bulk of both screens and reports were in the high category - many indeed far beyond the lower boundary of the high category. This indicates that, at least for this application, the fixed low, average and high categories of FPA are unsatisfactory.

5.8.2 Calibration of VFPs

For the purposes of calibration and comparison three calibration factors can be considered as shown below

	REFERENCE DATA		
	Functional partial size		Total
	Phase1	Phase2	size
BLOC	28100	29563	~32519
FPs	2575	2575	2575
b(BLOC/FP)	10.9	11.5	12.6

Table 5.36 Calibration Factors for Phases

For general calibration purposes, a *b* value of 12.6 is most appropriate, in view of the intent of FPA to estimate the total system size. The other *b* values are used for comparative purposes in 5.9.

The calibration table above illustrates a major difference between the technology of this chapter (technology B) and FPA (technology A). FPA estimates total size from summed and scaled **component contributions** which can be adjusted up or down depending on factors which may not influence the size of individual components at all. It does not measure component sizes directly, which seems unnatural for a method based on component size summation. However, for comparative purposes, this difficulty can be overcome by using the calibration factors of Table 5.36 for the Phase1 and Phase2 partial

sizes, thus enabling the FPA component contributions to be compared directly with the technology B component size estimates.

5.9 COMPARISON OF THE VFP MODEL WITH FPA

This section aims to answer the question "Is the model instance developed in this chapter, the VFP model, better than FPA when applied to the test data used here?"

The approach taken to answering the question is as follows

(i) the test data (a stratified sample) is scaled up by the relevant sampling factors for the strata (the component types) to obtain estimates of the actual test data population (test total LOC*). Table 5.37 shows the numbers of components in the sample strata in column 1, the number of components in the population strata in column 2, and the test total LOC* estimates in column 3. The sample is random and of sufficient size that test total LOC* must be very close to actual.

	1 Test sample	2 Test total	3 Test total LOC*	4 BLOC* phase1	5 % actual	6 BLOC* phase2	7 % actual	8 FPs	9 ALOC* phase1	10 % actual	11 ALOC* phase2	12 % actual
menus	5	5	247	247	1.00	247	1.00	15	164	0.66	173	0.70
relations	46	114	1594	1578	0.99	1578	0.99	865	9429	5.92	9948	6.24
reports	41	410	54120	51414	0.95	51955	0.96	2600	28340	0.52	29900	0.55
screens	41	82	5228	5176	0.99	5124	0.98	508	5537	1.06	5842	1.12
updates	28	56	1512			1331	0.88	0			0	0.00
partial size 1			61189	58415				3988	43469			
partial size 2			62701			60234		3988			45862	
% of actual				0.95		0.96			0.71		0.73	

Table 5.37 Comparison of the Model with FPA

(ii) the BLOC* estimates for Phase1 and Phase2 are obtained from the technology B model developed using the reference data whose estimation equations are summarized in Tables 5.21 and 5.22. These are compared with 'actual', i.e.

test total LOC*, both by component type and in total. Columns 4-7 show that the BLOC* results using the VFP model are quite good.

(iii) The FP values, using FPA, for each component class are shown in column 8. Again, these are estimates from the stratified sample and must be very close to actual.

(iv) Using the Phase1 and Phase2 calibration factors of 10.9 and 11.5 respectively, in reverse, the FPA function points in column 8 can be turned into equivalent lines of code, ALOC* at Phase1 and Phase2. The percentages of actual (column 3) are shown in columns 9-12.

Given the differences between the VFP model and FPA, it is necessary to go to some lengths to set up a fair comparison between the two. By adjusting them to equal Phase1 and Phase2 partial sizes for the reference data, the component contributions of FPA can be compared directly with the component size estimates of the VFP model.

It will be noted from Table 5.37 that, not only do the FPA component contributions not match the VFP model component sizes (i.e. their weightings are quite different) except for screens, but the ALOC* totals are also underestimated by large amounts, 29% and 27% at Phase1 and Phase2 respectively. This is a consequence of the different and inappropriate component weightings of FPA combined with the different population characteristics of the reference and test data sets. Specifically, in the test data set, the proportion of relations, which have by far the greatest weight in FPA, was about half that in the reference data set, whereas the proportion of reports, which have a low weight in relation to their LOC, was more than double that in the reference data set.

It is also instructive to compare the individual FPA component contributions with the corresponding individual VFP model component sizes within component type. This is difficult to do directly because the component type weightings are different. For example, there is a large systematic underestimation of ALOC* for reports on the part of FPA. In order to obtain a direct comparison the ALOC* values for FPA will be scaled so that in total they are equal to BLOC* values for the same component type. Having done this, the normal evaluation criteria of 5.5.2 are applied. Tables 5.38 and 5.39 show the results of such an intra-component type comparison for screens and reports.

SCREENS (ALOC* values scaled to same total size as BLOC* values)								
	n	RE*	MRE*	PRED(.25)	RMS	RMS*	LOC*	Actual
(VFP) BLOC*	41	-0.024	0.133	0.85	17.42	0.273	2570	2615
(FPA) ALOC*	41	-0.184	0.368	0.46	32.77	0.514	2570	2615

Table 5.38 Intra-Component Type Comparison of Screen Size Estimates

The evaluation criteria for the ALOC* derived from FP counts for the screens are clearly much worse than the BLOC* estimates. An examination of the individual values shows that the ALOC* values have a relatively small range (from 30 - 101) whereas the actual range of screen sizes is from 19 - 251. Thus the small screens tend to be overestimated by ALOC* (a negative RE) and the large screens tend to be underestimated (a positive RE), but relatively the large negatives predominate.

REPORTS (ALOC* values scaled to same total size as BLOC* values)								
	n	RE*	MRE*	PRED(.25)	RMS	RMS*	LOC*	Actual
(VFP) BLOC*	41	-0.018	0.190	0.71	46.92	0.356	5212	5407
(FPA) ALOC*	41	-0.491	0.781	0.10	99.24	0.752	5212	5407

Table 5.39 Intra-Component Type Comparison of Report Size Estimates

The evaluation criteria for the FP-derived ALOC* estimates for reports are also much worse. The reasons for this are similar to the reasons for the poor screen size estimates, but the situation is made much worse by the much greater size range of actual reports (from 24 to 468) compared to the low to high FPA/ALOC* range of 80 - 140. It is further aggravated by the fact that most of the reports (29 out of 41) were at the high FP level, with very few at the average or low level. The fixed FPA levels for low average and high outputs were inappropriate for this particular application.

In summary

(i) The VFP model instance here is remarkably robust in relation to this particular case where both the sizes and proportions of the component types were markedly different in the test population from those of the reference population.

(ii) The relative FPA component type weightings did not fit the technology B component types in terms of LOC required to implement them. As a result individual component sizes are poorly estimated (relations overestimated by 600%, reports underestimated by 50%). In addition, changing proportions of component types in a population can cause gross estimation errors.

(iii) The low, average and high classifications of FPA did not fit the data in this case. Specifically most of the screens and reports were in the high category, with relatively few being average or low. This can be seen from the average FP count for a report of

$$260/41 = 6.3$$

whereas the standard high, average and low counts are 7, 5, 4 respectively. However, having said all this, providing considerable care is exercised, FPA can be used, in the absence of anything better, as a basis for initial size estimation. [VERN88] describes the successful use of FPA for the estimate of the size of the first increments of the Correspondence School system, from which the reference and test data used here were later obtained.

(iv) The restricted range of FP weights for low, average and high components (3, 4, 6 for inputs; 4, 5, 7 for outputs) is far too narrow to effectively represent the size ranges of actual screens (which, including inquiries, covers both ranges) or reports in the development technology investigated here.

5.9 SUMMARY AND CONCLUSIONS

This chapter presents a detailed case study of the application of the model of Chapter 4 to a substantial body of data collected in one particular organization over a period of almost

four years. The administrative computer system within this organization represents a specific software development technology influenced by many factors, including

the general application category (data-centred, business)

special application area aspects (educational exceptions, varied and complex data requirements, classification and processing)

the 4GL technology employed

the development methodology (data-centred, incremental)

standards, management, policies, etc.

The administrative system is quite large (well in excess of 100,000 of 4GL code, equivalent to something over 800,000 lines of COBOL code) and the data samples on which the model instance is based and tested are themselves substantial (reference data 29,153 LOC, test data 9709 LOC).

The case study is thus a realistic application of the model in a modern development environment.

The data centred business application category is characterized in 5.2.

This category is further specialized for the particular software development technology of interest by setting up two component partitions, one during the specification phase (Phase1) and one during the early design phase (Phase2), as follows

Phase1	**Phase2**
menus	**menus**
MMENUS	
SCRMENUS	SCRMENUS only
relations	**relations**
screens	**screens**
reports	**reports**
all	all
non-ststistical	non-statistical
statistical	statistical
	updates
	batch file-to-file operations

The process of setting up vectors of candidate estimation variables is described in detail, including statistical and general criteria for selecting the most suitable estimation equations.

Some of the issues involved in splitting a diverse component type are examined in the particular case of reports, where two distinct populations, non-statistical and statistical reports were present. In this instance, it is debatable whether splitting the report component type into two is justified. On balance it is probably not worthwhile in this case, but it is an instructive exercise, nevertheless.

It is noted that no development environment is static and that, as a result, changes in the population of systems, subsystems or increments to be estimated, are inevitable with time. In this case such changes involved

(i) *relations*, where a bug in the development system led to a coding change and to a consequent reduction in LOC

(ii) *screens*, which were in general simpler in later increments, most of the very complex data entry screens having being completed early in the development

(iii) *reports*, which tended to become considerably more complex in later increments

(iv) large changes in the *relative numbers of components* of different types, there being many more reports, fewer relations and fewer screens in later increments.

These differences affect the test data in comparison with the reference data.

In spite of these population differences, which required changing only the prediction equation for relations, the model instance developed here showed remarkable robustness in its ability to predict the size of later increments, based on a stratified test sample taken from a large population (> 60,000 LOC).

A new function point measure is calibrated, with FPA as the reference sizing technology, using the methods of 4.4, after adjusting for the different basis on which FPA estimates final size from components which make up partial size.

The model instance (called VFP to distinguish it) is compared with what the traditional FPA method would have predicted. VFP gives very much better results in all respects than FPA. The comparison shows that the actual or implied technology on which FPA is based differs from the technology of the case study development and its size in LOC in the following respects

(i) FPA estimates total size from a different partial size, not including updates at Phase2

(ii) The weighting given to different component types by FPA is inappropriate

(iii) The fixed three-level, low, average, high, component ratings in FPA give both far too narrow a range and, in some cases, for example, reports, are offset from the real average in the case study technology.

Obtaining good size measures for effort estimation and scheduling is essential, both for systems as a whole and also for subsystems and increments within an on-going development environment.

This study shows convincingly, for this particular case, that a sizing model instance, developed from the generic model of Chapter 4, can give very good sizing estimates. These estimates are much better than FPA estimates and also better than conventional sizing wisdom would suggest. This is not surprising for a model that builds the characteristics of the software development technology into its estimation equations. The study also indicates, however, that there is a limit to the accuracy of prediction in the face of inevitable changes over time in the software technology used. Specifically, too precise a model may be over-sensitive to a drift in the characteristics of the population which is the subject of estimation.

An overall conclusion to this study is that the construction of a successful sizing model instance is dependent on real knowledge and experience of both the application area and the technology used to implement it.

CHAPTER 6

APPLICATION OF THE MODEL TO OTHER TECHNOLOGIES

This chapter describes three further examples of the application of the generic sizing model to a standard data-centred business system implemented using three different software development technologies. The framework of chapters 4 and 5 is followed but in significantly less detail. The statistical techniques used in chapter 5 are also used although again less detail is included. The three examples show the application of the model to develop prediction equations for corresponding component partitions for each of the different technologies corresponding to the Phase1 (or software requirements review) partition of Chapter 5. The purpose is to demonstrate that the model is applicable to several quite different software development technologies which can be used with the class of data-centred business applications.

6.1 SOURCE DATA

The data used in this chapter is the result of three different implementations of a subset of the standard IFIP inventory control and purchasing application [IFIP88]. These three systems have been developed using software technologies that are different from each other and also different from the software technology described in Chapter 5. The three systems used in this chapter were developed for use in a Master's thesis on a different topic by a graduate student with several years' commercial programming experience. Although the inventory systems will not be used commercially, they cannot be considered to be 'student programs' in the normal sense. The systems range in size from nearly 1500 LOC for the Advanced Revelation implementation to over 5000 LOC for the Micro Focus COBOL implementation. Table 6.1 summarizes the number of components, n, and total LOC in each of the systems analysed.

In lines of code, using Boehm's classification [BOEH81 p.65] this application is of small to intermediate size in terms of a conventional language, such as COBOL (5127). The

implementations in the two fourth generation languages are in the small range (1422 in Advanced Revelation and 2116 for Informix 4GL).

Software Technology	n	LOC
MIcro-focus COBOL	27	5127
Informix 4GL	27	2116
Advanced Revelation	33	1422
Totals	87	8665

Table 6.1 Source Data by Software Technology

6.2 SOFTWARE DEVELOPMENT TECHNOLOGY

A software technology includes the development methodology used together with the programming language and environment (see section 4.3.1). The three inventory system implementations described in this chapter were developed using entity-relationship modelling [CHEN77] and structured systems analysis [GANE79] followed by a modified state transition-based design technique for interactive systems [BEIL81]. All three implementations are based on the same data model, set of data flow diagrams, data dictionary and state transition diagrams. Any differences in implementation relate to the programming languages and environments used in the development of the three systems. The development strategy was that of incremental development. The languages used were Micro Focus COBOL Level II [MICR83], and two 4GLs, Informix 4GL [INFO87, INFO87a] and Advanced Revelation (AREV) [COSM87, COSM87a, COSM87b]. All three systems were developed on a large microcomputer under MS-DOS [MICR86]. The main characteristics of each of the languages are briefly discussed later in 6.4.1 - 6.4.3.

6.3 PRIMITIVE SIZE MEASURE

Once again LOC was chosen as the primitive size measure. This choice was made for the same reasons that it was chosen in the previous chapter. The counting convention used was that for LOC1 (physical lines) described in 5.4.1 with no blank lines, comment lines or JCL included.

6.4 THE COMPONENT PARTITIONS

Within each of the software development technologies a component partition is investigated which can be considered to be at the same stage in the development process as the Phase1 partition discussed in 5.6.1. In the case of AREV a Phase2 (preliminary design) partition is also briefly considered. For the other languages, additional design information available at Phase2 for this application is not easily related to countable program structures, so there is no difference between the Phase1 and Phase2 partitions.

6.4.1 Advanced Revelation

Advanced Revelation (AREV) [COSM87, COSM87a, COSM87b] is a 4GL which runs under an augmented Pick-like operating system which in turn runs under MS-DOS [MICR86] on the microcomputer used for the implementation. AREV provides powerful screen and form painting facilities linked to a data dictionary as well as features for handling menus and file enquiries. It uses a large number of active function key combinations to pass from one system-supplied design window to another. AREV also provides a procedural programming facility. Programs, subprograms or statements can be written in an extended, structured BASIC and can be embedded within forms or windows to meet more complex processing requirements than the standard facilities provide. AREV's procedural language, R/BASIC, can also be used to provide the structure for AREV transaction programs which call non-procedural elements when and if required.

An examination of the characteristics of the implemented functions, or modules, revealed that the appropriate component types for the software development technology were forms, pop-ups, menus and file listings. This process involved an iterative step before forms and windows were combined into one forms component type.

Forms. This component type includes forms and windows both of which are screen-based user interactions using a screen painter linked to a data dictionary. Windows are primarily input or update and forms primarily output, but both use essentially the same mechanisms. Most reports are implemented as forms in this technology. No file or

relation component is included explicitly in the AREV component partition since files are normally generated as by-products of forms, and all were generated in this way in this case. Counting them again separately would amount to double counting.

Pop-ups are system-generated control key-activated windows presenting lists of designated record attributes from a single file from which a selection of one or more records from that file can be made.

Menus are ring menus in a standard format with a help message displayed corresponding to the currently highlighted choice.

File listings are sorted listings, normally of selected records from one file. The listing may include some or all fields of all records of a file, or some or all fields of a fixed selection of the records of one file, together with standard headings, footings and some simple totals. Some fields may contain references to fields in other files which may contain similar references, etc., such references being dictionary-specified. File listings can normally be programmed in a handful of lines (anything between about 2 and 8 80-column lines with an average of 5). Only three examples of this component type were available in this case with sizes 4, 5 and 5. Because there are so few of them and their values are similar, the fixed value of 5 LOC has been included (as shown in Figure 6.1) for this component type.

The data used in the analysis comprised the following components

Component	n	LOC
menu	5	42
form	16	1291
pop-up	9	75
listing	3	14
totals	33	1422

Table 6.2 AREV Components

The prediction equations shown in Table 6.3 were obtained after analysis of the data. The second prediction equation for forms, including numbers of logics (ie. called BASIC subroutines) represents a Phase2 partition (preliminary design) in which additional

information concerning processing steps is available which can be related to countable program structures (subroutines). The predictions for both menus and pop-ups were exact hence the F-ratios and t-ratios are shown as ∞ in this table. As can be seen, though the R^2 value is rather low for forms at Phase1 (equation (1)), the other evaluation criteria indicate a good fit to the data. The Phase2 prediction equation (2) is, however, rather better.

	FORMS
1	LOC* = 41.03 + 9.47 data elements + 1.24 relations
2	LOC* = 49.33 + 6.38 data elements + 9.62 logics
	MENUS
3	LOC* = 2 + 1 choice
	POPUPS
4	LOC* = 6 + 1 data elements

	n	mean	R^2	RE*	MRE*	Pred(.25)	RMS	RMS*	F-ratio	t-ratios	s	df	p
1	16	80.7	41.8	-0.023	0.135	0.81	13.94	0.173	4.67	3.02, 2.39, 0.28	15.5	13	0.050
2	16	80.7	55.6	-0.025	0.126	0.88	12.17	0.151	8.15	4.01, 1.87, 2.04	13.5	13	0.015
3	5	8.4	100	0	0	1	0	0	∞	∞	0	4	0.000
4	9	8.3	100	0	0	1	0	0	∞	∞	0	8	0.000

Table 6.3 Prediction Equations and Evaluation Criteria for AREV

6.4.2 Informix 4GL

Informix 4GL [INFO87, INFO87a] has rather different characteristics from Advanced Revelation though it is also described as a 4GL. It has developed from a relational data

base management system which has been enhanced to include an SQL implementation and a complete high-level (or fourth generation) language, among other features. Special facilities include screen-building and menu-building utilities and a report writer. The Informix 4GL programming language can be described as conventionally procedural, but at a higher level (i.e. requiring fewer statements) than languages like COBOL or BASIC. Program units written in C can be called for more complex processing if required but the generality of the Informix 4GL programming language for normal data processing made this unnecessary in this particular case.

After some investigation it was found that a suitable component partition for the Informix 4GL implementation was into the following component types:

menus
input/updates
reports/inquiries.

The term update in this context is used in a different sense than in 5.6.2. Here it refers to a user interaction which changes existing data base information (an input, on the other hand, adds new information). The numbers and sizes of the components making up the Informix 4GL data set are shown in table 6.4.

Component	n	LOC
menu	5	134
input/update	11	1070
report/inquiry	11	912
Total	27	2116

Table 6.4 Informix 4GL Components

It is important to note that the component partitions described above are based on this particular, limited case. A more extensive history data base may indicate the need for some modification of these partitions. It should also be noted that the partitions were only arrived at after careful examination of the components characteristics and several partitioning iterations.

The prediction equations fitted to the data and the evaluation criteria for these equations for the Informix 4GL data set are shown in Table 6.5.

INPUTS/UPDATES
LOC* = 54.7 + 8.47 data elements + 6.26 relations

REPORTS/INQUIRIES
LOC* = 56.5 + 0.27 data elements + 12.7 relations

MENUS
LOC* = 4.96 + 3.41 choice

n	mean	R^2	RE*	MRE*	Pred(.25)	RMS	RMS*	F-ratio	t-ratios	s	df	p
11	97.3	0.59	-0.017	0.108	1	11.59	0.119	5.86	4.07, 2.14, 1.42	13.6	8	0.025
11	82.9	0.67	-0.015	0.093	0.91	10.14	0.122	8.19	4.35, 0.06, 3.03	11.9	8	0.010
5	26.8	0.92	-0.004	0.053	1	1.43	0.052	33.4	1.28, 5.78	1.79	3	0.005

Table 6.5 Prediction Equations and Evaluation Criteria for Informix 4GL

The Informix 4GL menus are shown in Table 6.5 as Pred(.25) = 1 They are, in fact rather better than that and are predicted within 10% 100% of the time. The evaluation criteria show a very good fit of the prediction equations to the data.

6.4.3 Micro Focus Level II COBOL

Micro Focus Level II COBOL [MICR83, MICR83a] is an extended COBOL which includes features intended to provide increased productivity over standard COBOL in a PC environment. In particular, it includes a powerful screen-painting facility, Forms-2, which substantially reduces the programming required for screen-based inputs and outputs.

The component partition for the Micro Focus COBOL development environment was found to be the same as that for Informix 4GL, namely:

menus
inputs/updates
reports/inquiries.

This partition was the result of combining some component types because of their strong similarities. Table 6.6 summarizes the numbers, n, and sizes of the components in the data set analysed.

Component	n	LOC
menu	5	445
input/update	11	2280
report/inquiry	11	2402
Total	27	5127

Table 6.6 Micro Focus COBOL Components

The prediction equations for Micro Focus Level II COBOL, and their evaluation criteria, are shown in Table 6.7.

INPUTS/UPDATES
LOC* = 105.0 + 23.4 data elements + 9.74 files

REPORTS/INQUIRIES
LOC* = 75.8 + 6.71 data elements + 58.5 files

MENUS
LOC* = 68.13 + 3.26 choice

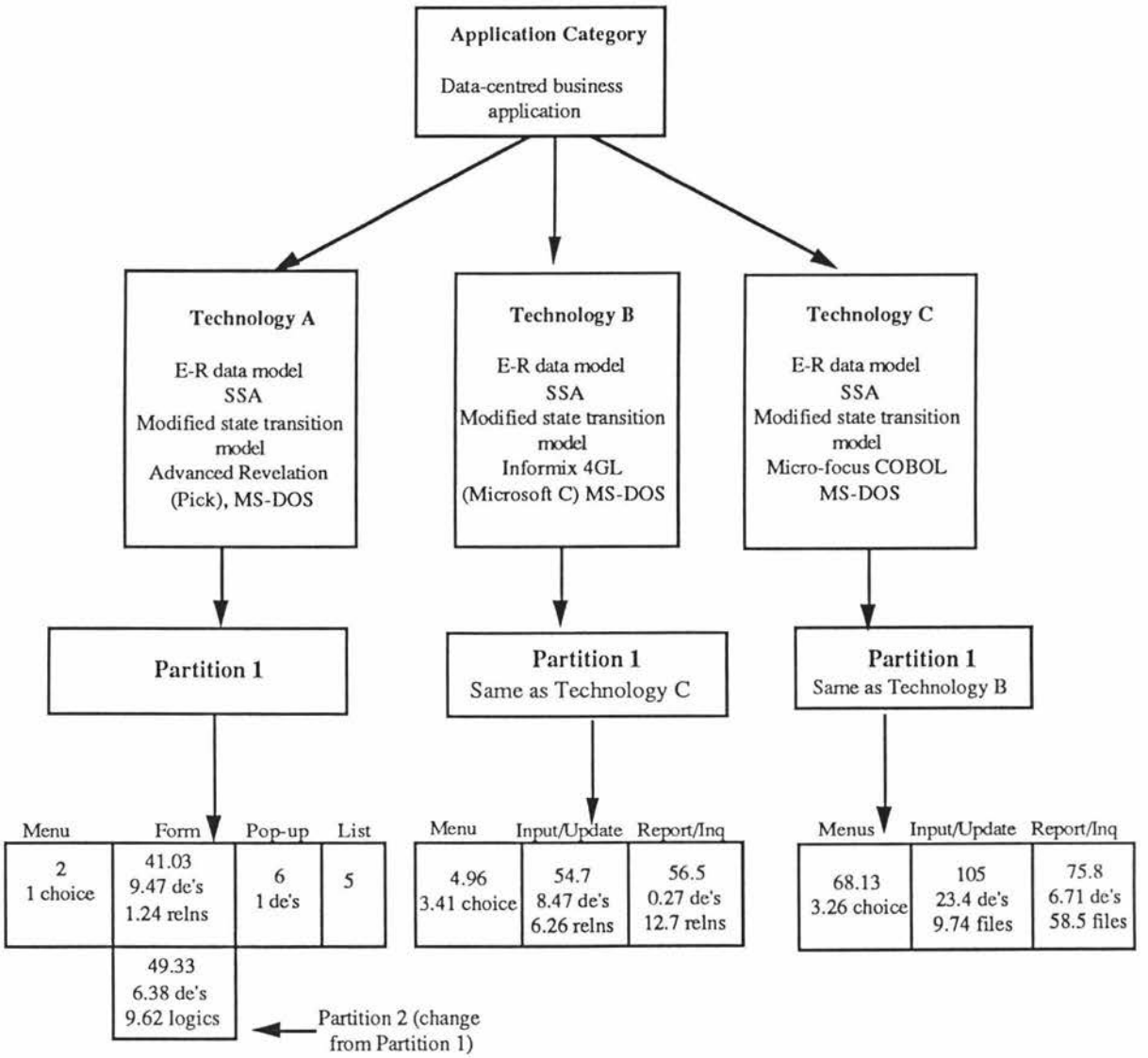
n	mean	R^2	RE*	MRE*	Pred(.25)	RMS	RMS*	F-ratio	t-ratios	s	df	p
11	207	0.48	-0.031	0.159	0.91	33.98	0.164	3.75	2.67, 2.01, 0.76	39.9	8	0.065
11	218	0.91	-0.012	0.086	1	22.12	0.101	41.3	2.68, 0.73, 6.40	25.9	8	0.001
5	89	0.91	-0.001	0.015	0.75	1.43	0.016	28.8	17.2, 5.37	1.84	4	0.010

Table 6.7 Prediction equations and Evaluation Criteria for Micro Focus COBOL

Though the reports/inquiries are shown as being predicted within 25% 100% of the time. They are in fact rather better than that being predicted within 17% 100% of the time. The evaluation criteria show a good fit to the data set.

6.5 COMPARISON OF THE SIZING MODEL INSTANCES FOR THE THREE SOFTWARE DEVELOPMENT TECHNOLOGIES

Figure 6.1 gives a summary of the structure of the three sizing model instances showing their development methodologies, languages, component type partitions and variable vectors, with estimation coefficient values, at the leaves.



de's = data elements, relns = relations

Figure 6.1 Summary of Sizing Model Instances for Three Software Development Technologies

It will be noted that, in spite of the wide differences in the languages and their environments (from a compact, powerful Pick-based 4GL to a conventional, though enhanced, DOS/COBOL environment), the component partitions have similarities. In fact they are the same for technologies B and C. By comparison, technology A includes

all input/update components and some report/inquiry components under its form component type. Other report/inquiry components are implemented either as pop-ups or as file listings. There are also some additional pop-up components with no equivalent components at all in the other two technologies.

There are also similarities between these component partitions and the Phase1 component partition of 5.6.1, whose menu, screen and report component types correspond as follows with the technology B and C component types of this chapter:

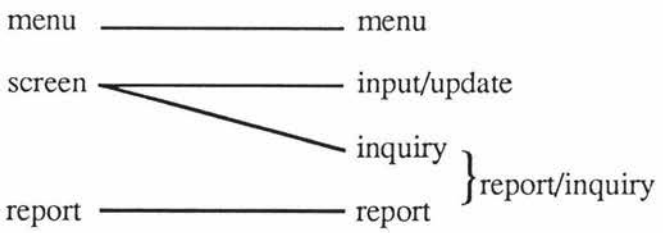


Figure 6.2 Component Partition Correspondences

None of the technologies of this chapter explicitly specifies a separate data model or schema, relations being included in one or more of the designated component types. As noted in 5.8.1 and Figure 5.2, these component partitions have some similarities with the traditional FPA component partition into files, inputs, outputs, inquiries (and external interfaces, not used here).

The similarities of the component partitions in Chapters 5 and 6 are considered to be evidence of the underlying and pervasive influence of the application category, namely data-centred business systems. It is suggested that component types like these are typical of implementations of systems in this application category, and correspond to some natural features of such systems. The differences in the component partitions on the other hand, are indicative of the differences between the languages employed to implement systems within this application category.

A corollary of these observations is that the FPA component partition has no special claims to be technology-independent or language-independent. On the contrary, it would appear to characterize some implied, perhaps composite, batch processing-influenced

technology (with its separation of input and output component classes) whose further details are obscure. As such it should have no special status except as *de facto* reference technology.

6.6 SUMMARY AND CONCLUSIONS

In this chapter three different software development technologies, largely characterized by their languages (Advanced Revelation, Informix 4GL and Micro Focus Level II COBOL) are applied to the standard IFIP inventory control and purchasing system. Though of the same general application class, data-centred business systems, this system is a very much smaller, simpler and more straightforward case study than that of Chapter 5. Nevertheless, it is a significant and not atypical application of its general class. It is also of a size that takes it out of the toy, student program class.

The application of the generic model to the implementations in the three different technologies was straightforward. However, a good knowledge of the technologies, their strengths and natural structures, and their relationship to the data modelling/structured analysis menu-based methodology used, was essential for the clear definition of component types.

The three technologies have significant differences. Their products have large size differences (a ratio more than 3:1 in the case of Micro Focus COBOL and AREV). They have different environments and philosophies, ranging from a Pick-like environment with 20-50 active function key combinations and many automatic tools to a conventional DOS/COBOL environment, admittedly with some significant extensions. It is therefore considered that the three technologies are diverse enough to provide a good indication of the range of applicability of the model, at least within the data-centred business application category.

The similarities and differences of the component partitions for the technologies of this chapter, that of Chapter 5, and FPA are analysed, revealing both significant similarities and significant differences. It is suggested that the similarities are characteristic of the common application category and the differences characteristic of the different languages used. It is further suggested that the FPA component partition has no special claims to be technology-independent or language-independent. As such it should have no special

status except as a *de facto* reference technology in common use. The three model instances derived in this chapter each have a close fit to the data. The statistical evaluation criteria in each case are also very good.

The study in this chapter therefore serves to increase one's confidence in the conceptual soundness and general applicability of the generic model.

CHAPTER 7

SUMMARY, CONCLUSIONS AND FURTHER WORK

It is not the purpose of this chapter to give a complete summary of the work of this thesis. That purpose is served by the summaries given in the final sections of Chapters 2-6. The aim of this chapter is firstly to place this work in software sizing in context and secondly to briefly identify those topics which the author believes to be the major contributions of the thesis.

7.1 THE CONTEXT OF THIS WORK

The research into software sizing reported in this thesis has been done in a context which is summarized briefly below. Some understanding of the background is necessary for the significance of the work to be appreciated.

(i) There is comparatively little research into software size estimation, except perhaps in the areas of large real-time and embedded military systems. In particular, there is very little work in the area of business systems and most of what there is does not address modern business software development environments, such as fourth generation languages.

(ii) The literature on software size estimation is very thinly scattered through a variety of journals, conference proceedings, departmental reports and supplier manuals, without any single major concentration of interest. There are no accepted standards for units of size measure, for the contents of sizing history data bases, nor for the classification of application categories, except perhaps in some very specific defence situations. It is perhaps not surprising, therefore, that the classification of software methods and metrics prior to this thesis has been fragmentary, incomplete and generally unsatisfactory.

(iii) Most current software sizing methods are aimed at the prediction of final product size in LOC. A number are proprietary products, the details of whose internal workings are a

carefully guarded secret. A few methods, notably the FPA family, are intended for productivity studies as well as final product sizing, though the concept of software productivity is often not clearly defined.

(iv) At the present time, software size estimation, no matter what method or combination of methods is used, gives results of very mixed reliability. For example, in a recent controlled test of 6 automated models, variations from a 28% underestimate to an overestimate of more than 300% were observed.

(v) For systems measured in function points, not only is subjectivity a problem, but there is also no objective way by which estimates or measurements can be compared with actuals. For all that function points are commonly used, what they are still remains something of a mystery.

7.2 MAJOR CONTRIBUTIONS OF THIS THESIS

Paragraphs 1 - 7 below summarize those topics which are the major contributions of this thesis to software size estimation research. These brief summaries should be considered within the context of 7.1.

1 A Classification of the Purpose of Software Sizing

There has been some confusion concerning the purpose of software sizing. Some software sizing studies are now made for purposes other than traditional final product size estimation. In particular, studies of developer productivity and technology productivity are becoming common, the latter assuming growing importance as new technologies emerge and attempts are made to compare developments that use them. The sizes required for different purposes are themselves different in kind. These topics are examined and clarified initially in Chapter 1 and thereafter throughout the thesis.

2 The Separation of Software Size Metrics and Methods

The author has observed, and indeed experienced, the difficulties of classification that result from a single combined treatment of software size metrics and software size estimation methods. Separate treatments of these related matters in Chapters 2 and 3

allow new classifications to be used that provide new insights, particularly into sizing methods.

3 A Survey and Classification of Software Size Metrics

A structural classification of software size metrics into textual metrics, object counts, vector metrics and composite metrics is given in Chapter 2 and a distinction is drawn between primitive metrics such as LOC, and derived metrics, such as function points. While the categories themselves are in current use in several different contexts, the structured treatment of the topic is new. Within this structure there is a definitive survey of software size metrics.

4 A Survey and Classification of Software Size Estimation Methods

A new structural classification of software sizing methods is presented which provides the main framework for Chapter 3. This classification relates sizing methods to a top-down structural decomposition of the software to be sized and at its leaves includes the more traditional approach-based classification. Besides being a logical and effective classification, it also contributes some essential features to the generic sizing model developed in Chapter 4.

5 The Development of a Generic Software Size Estimation Model Based on Component Partitioning

A generic software size estimation model is developed, based on component partitioning, from which a wide range of sizing model instances can be derived. The model subsumes many structurally similar approaches and is both a generalization and extension of existing methods which use component partitioning of one kind or another, in particular FPA-like methods. The following objectives which were initially set out in section 1.3 (4) are achieved by the generic model:

(1) accommodates a wide range of different sizing purposes and sizing metrics,

(2) overcomes many of the criticisms of existing models,

(3) spans much of the software life cycle, but concentrates on the specification and preliminary design phases where early size estimates are much in demand,

(4) can be based on more objective aspects of software representations from specification through to code, but does not necessarily exclude more subjective aspects,

(5) distinguishes metrics which are highly technology-dependent from those that are less so, relates them through technology-dependent factors, and provides a calibration mechanism for metrics with low technology-dependence (job-size metrics),

(6) builds upon a historical data base of technology-dependent data and includes an adaptive mechanism to cope with a shift or change in software technology, including recalibration of job-size metrics where necessary,

(7) can be tailored to specific development environments or technologies for greater accuracy, or can be kept more general in applicability, possibly at some cost in accuracy,

(8) allows for partial sizing to meet a variety of sizing purposes, and

(9) includes an adjustment factors model which characterizes and classifies adjustment factors effectively and provides for flexibility in their use to meet different sizing purposes.

6 The Derivation and Testing of a Realistic Model Instance

A model instance is developed in detail in the context of a large and complex system of data-centred business applications. The construction of the model instance provides not only an example of the application of the generic model, but also numerous insights into the nature of the model and the ways in which it takes advantage of the characteristics of the particular software development technology used to obtain better estimates and measures. The model instance is created from a substantial body of reference data, and then applied to a later sample of test data. The performance of the model is excellent at the system level (within 4%) and also meets or exceeds most statistical evaluation criteria at the individual component level (for example, within 25% of actual 75% of the time). The model instance is compared with FPA for this particular case and is shown to be very much better. This comparison also reveals something of the anatomy and

technology dependence of FPA and shows how inappropriate the FPA component type weightings are for the software development technology used in this application.

7 The Applicability of the Model to Other Technologies
The applicability of the generic model is further demonstrated by its successful application to three different software development technologies which are used to implement a subset of the standard IFIP inventory and purchasing application. These technologies vary from a conventional COBOL PC environment to a highly-specialized Pick-based 4GL environment. The three model instances developed provide excellent fits to the available data. Some observations are made on the relative influence of the application category and the implementation technology on the sizing model.

7.3 GENERAL CONCLUSIONS

Following on from the specific contributions highlighted in the previous section, some more general conclusions can be drawn from this work:

1. Structural approaches to the classification of software size estimation methods and software size metrics can provide new insights into both methods and metrics.

2. Models which recognize and take advantage of technology dependence in the sizing process can provide more accurate technology-dependent product size measures; they can also give better technology-independent measures which allow for, and remove, most programming language/environment dependencies and are calibrated to be comparable with established function point measures.

3. The instantiations of the generic model developed in this thesis demonstrate remarkable accuracy and robustness across several different software development technologies and in the face of substantial changes over time in the characteristics of the population which is the subject of estimation; this gives some confidence in the applicability of the generic model to data-centred business applications, though its applicability may in fact be much wider than this.

7.4 FUTURE RESEARCH WORK

Future areas for research related closely to this thesis are briefly outlined below. In considering these areas it is important to remember that the main difficulties in software size estimation research are related to the availability, variety and quality of data, including an understanding of the environments in which it has been collected. To obtain and analyse a significant body of data from a variety of environments normally requires a sustained team effort, including industry participation, over a minimum time span of several years.

1. The application of the model to other application categories and software development technologies. This is a major task involving co-operating organizations and a thorough knowledge of the application categories and technologies used.

2. The investigation of specification sizing metrics. For technology productivity studies a technology-independent measure of the size of the job to be done is of considerable interest. The use of CASE tools and formal or semi-formal specification languages are relevant in this type of study.

3. The investigation of sizing, and its possibly changing relation to effort estimation, in the emerging CASE environments, through suitable instrumentation of CASE tools.

4. A study of system adjustment factors. This is again a major study concerned with the purposes of sizing, the sizes required, and the technologies involved. An investigation embracing a number of different co-operating organizations would be necessary to establish a sufficiently broad base of study.

5. The development of proposals for standards for data collection for history data bases for sizing, costing, scheduling and productivity data.

The general goal of future work in this area is a better understanding of all the factors affecting software size in order that better conceptual models may be developed leading to the availability of widely-applicable sizing components within comprehensive costing, scheduling and project management tools.

REFERENCES

ALBR79 A. J. Albrecht, "Measuring Application Development Productivity," Proceedings Joint SHARE/GUIDE/IBM Application Development Symposium, Oct 1979, pp. 83-92 also reprinted in [JONE81] pp. 34-43.

ALBR83 Allan J. Albrecht and John E. Gaffney, Jnr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983, pp. 639-648.

ALBR84 A. J. Albrecht, AD/M Productivity Measurement and Estimate Validation - Draft, IBM Corp. Information Systems and Administration, AD/M Improvement Program, Purchase, NY, May 1984.

ARTH85 Lowell Jay Arthur, Measuring Programmer Productivity and Software Quality, John Wiley & Sons, 1985.

ARON69 J. D. Aron, "Estimating Resources for Large Programming Systems", NATO Science Committee, Rome, Italy, October, 1969.

BAIL81 John W. Bailey and Victor R. Basili, "A Meta-model for Software Development Resource Expenditures", Proceedings of the 5th International Conference on Software Engineering, California, March 1981, pp. 107-116.

BASI83 Victor R. Basili, Richard W. Selby, Jnr., and Tsai-Yun Phillips, "Metric Analysis and Data Validation Across Fortran Projects", IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983, pp. 652-663.

BASI85 Victor R. Basili and N. Monina Panililo-Yap, "Finding Relationships Between Effort and Other Variables in the SEL", TR-1520, Computer Science Technical Report Series, University of Maryland, July 1985.

BASI88 Basili, Victor R. and Rombach, H. Dieter, "The TAME Project: Towards Improvements Oriented Software Environments," IEEE TSE Vol. 14, No. 6, June 1988, pp. 758-773.

BEHR83 Charles A. Behrens, "Measuring the Productivity of Computer Systems Development Activities with Function Points", IEEE Transactions on Software Engineering, Vol SE-9, No.8, November 1983, pp. 648-652.

BIEL81 P. Bieleski, "Flowcharting Revisited", New Zealand Computer Society Conference, 1981, pp. 123-139.

BLUM86 Bruce I. Blum, "Four Yrs of Experience with an Environment for Implementing Information Systems", The Journal of Systems and Software 6, 1986, pp. 261-271.

BOEH81 B. W. Boehm, Software Engineering Economics, Prentice-Hall, 1981.

BOEH82 B. W. Boehm, Foreword to [DEMA82] Controlling Software Projects.

BOEH84 B. W. Boehm, Software Engineering Economics", IEEE Transactions on Software Engineering, January 1984, pp. 4-21.

BOZO86 GJB Associates, 552 Marine World Pkwy #1202, Redwood City, CA 94065.

BOYD84 Robert E. Boydston, "Programming Cost Estimate: Is It Reasonable?", Proceedings of the Seventh International Conference on Software Engineering, Florida, March 1984, pp. 153-159.

BRIT85 R. N. Britcher and J. E. Gaffney Jnr. "Reliable Size Estimates for Software Systems Decomposed as State Machines", Proceedings of IEEE COMSAC85, October 1985, pp. 9-11.

BROO87 Frederick P. Brooks, "No Silver Bullet, Essence and Accidents of Software Engineering", IEEE Computer, April 1987, pp.10-19.

BRYA83 A. Bryant and J. A. Kirkham, "B. W. Boehm Software Engineering Economics A Review Essay", ACM Sigsoft Software Engineering Notes, Vol. 8 No. 3, July 1983, pp. 44-60.

CANN86 Richard G. Canning, "A Programmer Productivity Controversy", EDP Analyser, Vol. 24, No. 1, January, 1986, pp. 1-11.

CARD87 D. N. Card and W. W. Agresti, "Resolving the Software Science Anomaly", Journal of Systems and Software, 7, 1981, pp. 29-35, pp. 83-84.

CHEN77 Peter P. Chen, The Entity Relationship Approach to Logical Data Base Design, The Q.E.D. Monograph Series, Data Base Management, No. 6, Q.E.D. Information Sciences, Inc., 1977.

COLE82 Michael L. Coleman, "Measuring Software Development Productivity, " Technical Report No. 3888A, Aluminum Company of America, Feb. 1982.

CONT86 S. D. Conte, H. E. Dunsmore and V. Y. Shen, Software Engineering Metrics and Models, The Benjamin/Cummings Publishing Company, Inc., 1986.

COSM87 COSMOS, Inc. Advanced Revelation - Menus and Windows, Cosmos, Inc., USA, 1987.

COSM87a COSMOS, Inc. Advanced Revelation - Using Advanced Revelation, Cosmos, Inc., USA, 1987.

COSM87b COSMOS, Inc. Advanced Revelation - Technical Reference, Cosmos, Inc., USA, 1987.

DACS87 "A Descriptive Evaluation of Software Sizing Models", prepared for Headquarters USAF/Airforce Cost Centre, Washington DC 20330-5018, by IIT Research Institute, 4550 Forbes Boulevard, Suite 300 Lanham, Maryland 20706-4324, September 1987.

DEMA79 T. DeMarco, Structured Analysis and System Specification, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

DEMA82 T. DeMarco, Controlling Software Projects: Management, Measurement and Estimation, Yourdon Press, New York, 1982.

DEMA84 T. DeMarco, "An Algorithm for Sizing Software Products, " Performance Evaluation Review, Vol. 12. No. 2, Spring-Summer 1984, pp. 13-22.

DEMI60 W. E. Deming, Sample Design in Business Research, Wiley, New York, 1960.

DOTY77 Doty Associates Inc., Software Cost Estimation Study, RADC-TR-77-220 Rome Air Development Centre, Rome, N. Y., 1977.

DUNC88 Anne Smith Duncan, "Software Development Productivity Tools and Metrics", Proceedings of the 10th International Conference on Software Engineering, April 11-15, 1988 Singapore, pp. 41-48.

DUNS84 H. E. Dunsmore, "Software Metrics: An Overview of an Evolving Methodology", Information Processing and Management, Vol. 20, No. 1-2, 1984, pp. 183-192.

ELSH78 J. L. Elshoff, "An Investigation into the Effect of the Counting Method Used on Software Science Measurements", ACM SIGPLAN Notices, Vol. 13, 1978, pp. 30-45.

FEUE79 Alan R. Feuer and Edward B. Fowlkes, "Some Results from an Empirical Study of Computer Software," IEEE International Conference on Software Engineering, 1979, pp. 351-355.

FITS80 George P. Fitsos, "Vocabulary Effects in Software Science", Proceedings of IEEE COMSAC80, pp.751-756, 1980.

FITZ78 A. Fitzsimmons and T. Love "A Review and Evaluation of Software Science", Computing Surveys, Vol 10, March 1978, reprinted in [JONE81] pp. 44-59.

FREI79 F. R. Freiman and R. E. Park, "Price software model--version 3; an overview," Proceedings, Workshop on Quantitative Software Models, 1979, pp. 32-41.

GAFF81 J.E. Gaffney Jnr., "Software Metrics: A Key to Improved Software Development Management", Proc. Computer Science and Statistics 13th Annual Symposium on the Interface", Pittsburgh, 1981, pp.211-220.

GANE79 Chris Gane and Trish Sarson, "Structured Systems Analysis: Tools and Techniques", Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

GORD87 Gordon Group, Before You Leap User's Guide 1987.

HALS77 M. H. Halstead, Elements of Software Science, Elsevier, New York 1977.

HELM66 O. Helmer, Social Technology, Basic Books, New York, 1966.

IFIP88 Example A - Business Analysis and System Design Specifications for an Inventory Control and Purchasing System, IFIP Working Group 8.1, Case Study Specifications for Computerized Assistance During the System Life Cycle, pp.A-1 - A-33, September 1987 for CRIS '88 September 1988.

INFO87 Informix 4GL Version 1.10, User Guide, Informix Software Incorporated USA, 1987.

INFO87a Informix 4GL Version 1.10, Reference Manual, Volumes 1 and 2, Informix Software Incorporated USA, 1987.

ITAK82 M. Itakura and Akio Takayanagi, "A model for Estimating Program Size and Its Evaluation", Proceedings of the Sixth International Conference on Software Engineering, September 1982, pp. 104-109.

JACK75 M. Jackson, Principles of Program Design, Academic Press, London, 1975.

JEFF79 D. R. Jeffery and M. J. Lawrence "An Inter-Organisational Comparison of Programming Productivity", IEEE International Software Engineering Conference, Munich, Germany, 1979, pp. 369-377.

JEFF83 D. R. Jeffery, "Managing Programmer Productivity", Systems Colloquia Series, London School of Economics, Nov. 1983.

JEFF87 D. R. Jeffery "A Software Development Productivity Model for MIS Environments", Journal of Systems and Software 7, 1987, pp. 115-125.

JEFF88 D. R. Jeffery, C. C. Loo and G. C. Low, " The Validity, Reliability and Practicality of Function Points as a Measure of Software Size", Information Systems Research Report, Department of Information Systems, University of New South Wales, 1988.

JONE78 T. Capers Jones, "Measuring Programming Quality and Productivity", IBM Systems Journal, Vol. 17, No. 1, 1978, pp 39-63. Reprinted in [JONE81] pp. 9-33.

JONE81 T. Capers Jones, Programming Productivity-Issues for the Eighties; IEEE Press; Silver Spring. MD; Cat. No. EHO 186-7; 1981.

JONE86 T. Capers Jones, Programming Productivity, McGraw-Hill Book Company, 1986.

KAFU85 Dennis Kafura, "A Survey of Software Metrics," Proceedings ACM Annual Conference, Denver, Colorado, 1985, pp. 502-506.

KEME87 Chris. F. Kemerer, "An Empirical Validation of Software Cost Estimation Models", Communications of the ACM, Vol. 30, No. 5, May 1987, pp. 416-429.

KITC85 Barbara A. Kitchenham and N. R. Taylor, "Software Project Development Cost Estimation", Journal of Systems and Software, 5, 1985, pp. 267-278.

KWON87 Arnold W. Kwong, "Measuring Software Development Productivity", The CASE Report, Nastec Corporation, April 1987, pp. 1-4.

LASS81 J. L. Lassez et al., "A Critical Examination of Software Science", Journal of Systems and Software, 2, 1981, pp. 105-112.

LEVI86 Anany Levitin, "How to Measure Software Size and How Not To", Proceedings of IEEE COMPSAC, 1986, pp. 314-318.

LI87 H. F. Li and W. K. Cheung, "An Empirical Study of Software Metrics", IEEE Transactions on Software Engineering, Vol. S-E 13, No. 6, June, 1987, pp. 697-708.

LIM89 P. H. Lim, "A Comparative Study of Programming Language Expansion Ratios", Unpublished M. Tech. Thesis, Massey University, 1989.

LIST82 A. M. Lister, "Software Science - The Emperor's New Clothes?", The Australian Computer Journal, Vol. 14, No. 2, May 1982, pp.66-70.

LOW88 G. C. Low and D. R Jeffery, "Function Points in the Estimation and Evaluation of the Software Process", Information Systems Research Report, Department of Information Systems, University of New South Wales, To appear in : Proceedings of the Australian Software Engineering Conference ASWEC-88 Canberra, May 1988.

MAST85 Thomas F. Masters II, "An Overview of Software Cost Estimating at The National Security Agency", Journal of Parametrics, Vol. 5, No. 1, March 1985, pp. 72-84.

MCCA76 T. J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering Vol. SE-2, No. 4, 1976, pp. 308-320.

MICR85 ALL User's Reference Manual Version 1.0, Microdata Computers (Holdings) Limited, January 1985.

MICR83 Micro Focus Personal COBOL Version 1.1 Getting Started, Micro Focus Ltd., 1983.

MICR83a Micro Focus Level II COBOL Version 1.1 Language Reference Manual, Micro Focus Ltd., 1983.

MICR86 Microsoft® MS-DOS® User's Guide, Microsoft Corporation, 1986.

MOHA81 S. N. Mohanty, "Software Cost Estimation: Present and Future," Software - Practice and Experience, Vol. 11, 1981, pp. 103-121.

OTTE86 J. E. Otte, "Parametric Software Sizing Case Study" Proceedings of IEEE COMPSAC 1986, pp. 707-710.

PARI82 Girish Parikh, How to Measure Programmer Productivity, Q. E. D. Information Sciences, Inc., Massachusetts, 1982.

PUTN78 L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimation Problem," IEEE Transactions on Software Engineering, July, 1978, pp. 345-381.

PUTN79 Lawrence H. Putnam and Ann Fitzsimmons, "Estimating Software Costs," Datamation, 189-198 September; 171-178 October; November; 1979, pp. 137-140.

PUTN87 Lawrence Putnam, "Size Planner, An Automated Sizing Model", Presentation to the Third COCOMO User's Group Meeting Pittsburgh, November 1987.

QSM87 QSM Inc., McLean, VA 22101 "Software Investment Management", Presentation to the Third COCOMO User's Group Meeting, Pittsburgh, November 1987.

REIF87 Donald J. Reifer, "An Introduction to RCI's Resource Estimation Tools", RCI-TN-302, Presentation at the COCOMO User's Meeting, Pittsburgh, November, 1987.

REIF87a Donald J. Reifer, "SoftCost-R: User Experiences and Lessons learned at the Age of One", Journal of Systems and Software, 7, 1987, pp.279-286.

REIF87b Donald J. Reifer, personal communication to author, 1987.

RUBI83 H. Rubin, "Macro-estimation of software Development Parameters", Proceedings of SOFTFAIR '83, July, 1983, IEEE Press, pp.109-118.

RUBI85 Howard A. Rubin "A Comparison of Cost Estimation Tools - (A Panel Session)", Proceedings of the 8th International Conference on Software Engineering, London August, 1985, pp. 174-180.

RUDO83a E. Rudolph, "Productivity in Computer Application Development", Dept. of Management Studies, Working Paper no. 9, University of Auckland, March 1983.

RUDO83b E Rudolph, Measuring Software Development Productivity", Proceedings New Zealand Computer Society Conference Wellington, 1983, pp. 417-425.

SHEN79 V. Y. Shen "The Relationship Between Student Grades and Software Science Parameters", Proceedings 3rd International Conference on Computer Software and its Applications (COMSAC), Chicago, November 1979.

SHEN83 V. Y. Shen, "Software Science Revisited: A Critical Analysis of the Theory and it Empirical Support", IEEE Transaction son Software Engineering, SE-9(2) 1983, pp. 155-165.

SINC87 Brett Sinclair, Personal communication to the author.

SPR86 Software Productivity Research, Inc. (SPR), SPQR/20 User's Guide, 1986.

SYMO88 Charles R. Symons, "Function Point Analysis: Difficulties and Improvements," IEEE Transactions on Software Engineering, Vol., 14, N0. 1, January 1988, pp. 2-11.

TATE87 Graham Tate and June Verner, "4GLs-Expectation and Experience", NZ Computer Society Conference, Christchurch, August 1987, pp. 188-200.

TATE88 Graham Tate, June Verner and Richard Hayward, "A Proposal for a CASE System for Medium Sized Business Applications", Advance Working Papers, International CASE Symposium, Cambridge MA. USA, July 1988.

THAY81 R. H. Thayer, A. B. Pyster, and R. C. Wood, "Major Issues in Software Engineering Project Management," IEEE Transaction on Software Engineering, July 1981, pp. 333-342.

VERN87 June Verner and Graham Tate, "A Model for Software Sizing", Journal of Systems and Software, July 1987, pp 173-177.

VERN88 June Verner and Graham Tate, "Estimation of Size and Effort for a Fourth-Generation Development", IEEE Software, July 1988, pp. 15-22.

VERN89 June Verner, Graham Tate, Barry Jackson and Richard Hayward, "Technology Dependence in Function Point Analysis: A Case Study and Critical Review", to appear in Proceedings 11th ICSE Conference, Pittsburgh, May 1989.

WALS77 C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation", IBM Systems Journal, Vol. 18, No. 1, 1977, pp. 54-73. Reprinted in [JONE81] pp. 60-80.

WANG84 Andrew Wang, The Estimation of Software Size and Effort: An Approach Based on the Evolution of Software Metrics, Ph. D. Thesis, Purdue University, July, 1984.

WANG85 A.S. Wang and H.E Dunsmore, "Early Size Estimation: A Critical Analysis of the Software Science Length Equation and a Data-Structure-Oriented Size Estimation Approach", Computer Sciences Department, Purdue University, 1985.

WEIN74 G. M. Weinberg and E. L Schulman, "Goals and Performance in Computer Programming", Human Factors, Vol. 16, no. 1, 1974, pp. 70-77.

ZWAN84 K Zwanzig, Ed. Handbook for Estimating Using Function Points, GUIDE Project DP-1234 GUIDE Int., Nov., 1984.

GLOSSARY

4GL	fourth generation language (q.v.).
adjusted function points	raw function point value modified up or down by system adjustment factors, i.e. the information processing size adjusted for the degree of technical difficulty.
adjustment factors	factors applied to the raw function points or information processing size to account for the degree of technical difficulty involved in implementation. Currently there are 14 different factors in standard FPA [ALBR79, ALBR83, ALBR84] (see 3.2.3, 3.2.3 and 4.5).
algorithmic	describes a method that uses one or more well defined algorithms rather than, for example, expert judgement.
analogs	used in Price SZ (3.2.1 iii) to describe continuous variables generally input by sensor devices such as radar, temperature, or other real-time input devices [DACS87].
application category	a class of functionally and/or structurally similar systems (see 4.3.1).
ARCH	architectural constant to deal with the degree of centralization or distribution of the application and its computers. Used in ASSET-R [REIR87] (see 3.2.2B ii).
ASSET-R	Analytical Software Size Estimation Technique - Real Time [REIF87] (see 3.2.2 B ii).

availability profile — for each component partition and within this for each component type this profile may be constructed showing at what stages of the development process and for what purposes the particular component partition and candidate variable vectors are relevant and when the information they use is available (see 4.3.2).

BANG — a composite size metric [DEMA82], either System BANG, the weight of function to be delivered (based on functional primitives and input and output elements at their boundaries) or Data BANG (based on counts of objects in the data base and the number of their relationships) See 2.5.3.

basic measures — primitive measures.

Beta distribution — a two parameter family of probability distributions (refer to any standard statistical text).

BYL — Before You Leap [GORD87] an automated software cost estimation and scheduling model that uses function points and COCOMO.

candidate variables — see 4.3.2.

CASE tools — Automated tools to aid in software development called variously Computer Aided System Engineering or Computer Aided Software Engineering tools.

CDR — Critical design review. Takes place at the end of detailed design and is followed by implementation. see 3.4.

CEIS — Computer Economics, Inc. Sizer [DACS87] (see 3.2.2 A i d).

character count	a primitive metric based on counts of all single characters in an implementation. See 2.2.4.
COCOMO	COnstructive Cost MOdel, an automated cost and scheduling tool [BOEH81].
common metrics	measures that are widely applicable to many development methods and to many languages (see 2.6).
component	may include modules, structures, objects or other distinguishable and separately countable units within a system. (see 4.3.2).
component partition	a division of the components of a system at any phase into a mutually exclusive set of component types covering all the aspects of a system that are of interest at that phase of development (see 4.3.2).
composite metrics	single values produced by the application of functions to more than one kind of measure or count. They can be considered to be functions of vector metrics and are usually also derived metrics (See 2.5).
COPMO	COoperative Programming MOdel. A model for estimating project and team size [CONT86].
Curve fitting	see 3.2.2 A i b.
Delphi technique	group consensus technique (see 3.2.1).
derived metrics	metrics which are normally composite but in addition, are related to some other metric(s), either explicitly or implicitly, by some calibration process.

developer productivity	a measure of developer and/or managerial effectiveness within a defined software development environment (see 1.2.1).
df	degrees of freedom.
DSI	delivered source instructions (see 2.2.1 i).
DVAR	a count of the unique variables identifiable at the design stage (see 3.2.2 A ii a).
ESD	Electronic Systems Division (see 3.2.2 B i b).
EXPF	technology expansion factor. Used in ASSET-R [REIR87] (see 3.2.2B ii).
FAA	Federal Aviation Administration.
Feature Points	an extended FPA-based approach [SPR86] (see 3.2.2 B).
Fourth generation language	a commonly used term to describe a variety of modern high productivity application development systems.
FP	function point (q.v.).
FPA	Function Point Analysis.
function points	derived composite metric that attempts to measure function value that a system provides to a user. It uses files in combination with input and output data elements or record types as the basic counts from which function points are caluculated (see 2.5.1).

functional bulkiness	used in Price SZ [DACS87] to describe the effects of software team experience with language and availability of software tools. Normal values = 1 more tools < 1 and fewer tools > 1 (see 3.2.1 iii) .
functional primitives	used in the System BANG metric (see 2.5.3).
HIPO	Hierarchical Input Output analysis method.
information processing size	term used in FPA to refer to raw or unadjusted function points (see 2.5.1).
input analogs	used in Price SZ (3.2.1 iii); continuous variables generally input by sensor devices such as radar, temperature or other real time devices [DACS87].
I/O count	total program input/output data element count without the weights and processing complexity adjustment applied in function point analysis; used in [ALBR83] and treated there as being equivalent to Halstead's n^*_2 (see 3.2.2 B ii a)
JCL	Job Control Language.
KDSI	thousands of delivered source instructions.
KTOC	thousands of tokens.
line of code	see 2.2.1.
language expansion ratio	Normally used in FPA-based size estimation approaches where it is a ratio of lines of code per function point for specific languages (see Table 3.3). However it is used in Price SZ to describe the combination of languages and

	compilers used in the development effort. For size output in machine level instructions the value = 1.
length	a Software Science metric. The length of a well-structured program is considered to be a function of unique operators and operands $N = N_1 + N_2$ (see 2.5.4).
linguistic approaches	includes those size estimation approaches that count the lexical symbols used in the programmatic expression of an algorithm (see 3.3.4).
LOC	lines or line of code.
LOC*	estimated lines of code.
MARKII	a composite derived metric [SYMO88] based on FPA that uses input data elements, output data elements and entities for each logical transaction type as the basic syntactic counts from which function points are calculated (see 2.5.2).
metric vectors	see 2.4.
MLI	machine level instructions.
MRE	magnitude of the relative error (|RE|).
MRE*	mean magnitude of the relative error.
n_1	number of unique operators
n_2	number of unique operands.
n^*_2	estimate of n_2 (see 3.2.2 A i d). Input/output data item counts. The sum of the overall inputs and outputs of an application program (see 2.5.4 ii).

n	vocabulary metric ($n_1 + n_1$).
N_1	total numbers of operators.
N_2	total numbers of operands.
N	program size metric ($N_1 + N_2$).
^N	a Software Science metric, estimated length (N) (see 2.5.4).
non-executable lines	lines containing declarations and other non-imperative statements in programs.
object code size	size of compiled code in machine language instructions.
operand	a symbol used to represent data in a program.
operator	any symbol or keyword in a program that specifies and action, including a separator; any symbol that is not an operand.
partial sizing	summing of component size estimates (see 4.2).
PDR	Preliminary design review. Takes place at the completion of the preliminary design and is followed by detailed design.
PERT	Project Estimating and Reporting Tool (see 3.2.2 for use in size estimation).
PFR	Product feasibility review. Takes place part way through the feasibility study and is followed by the software requirements phase.

PRED(L)	prediction at level L.
primitive metrics	measures that are conceptually simple and can be counted directly at some stage in development (see 2.6).
QSM Size Planner	Quantitative Software Management Size Planner [PUTN87] an automated tool that includes three different approaches to software size estimation, Fuzzy Logic, Function Points and Standard Components Sizing (see 3.2).
R^2	coefficient of multiple determination.
raw function points	information processing size.
RE	relative error.
reference technology	a technology that has well-established and sufficiently reliable metrics often (function points) have been obtained in the past (see 4.3.6).
RMS	root mean square error.
RMS*	relative root mean square error.
scaffolding	also referred to as test harness. Non-delivered code used mainly to construct a testing framework during modular development.
SERC	Software Engineering Research Center.
SLOC	source lines of code.
size calibration factor	used in Price SZ [DACS87] to describe the effect of the user organization's specific know-how or the way the

organization implements software projects. Normal value = 1 with range expected from 0.7 - 1.5 (3.2.1 iii).

SIZER/FP — an FPA-based size estimation model [SPR86] (see 3.2.2. B i).

sizing by analogy — describes approaches relating the size of the proposed development to previously developed modules and systems of similar function and environmental characteristics (see 3.3.1).

SizeA — part of the general sizing and costing model presented in Chapter 4 (see 4.1).

SizeB — part of the general sizing and costing model presented in Chapter 4 (see 4.1).

SizeC — part of the general sizing and costing model presented in Chapter 4 (see 4.1).

Size-in Size-out — see 3.3.2.

SLOC — source lines of code.

SMM — State Machine Model [BRIT83] size estimation approach (see 3.2.2 i c).

software development technology - a combination of the application category, the development methodology (including its lifecycle model) and the languages or software engineering environments used in a particular development situation (see 4.3.1).

software economics — a general area comprising software size, cost, effort and schedule estimation, risk and value analysis, productivity studies, etc.

Software Science — a set of composite metrics [HALS77] based on counts of operators and operands (see 2.5.4).

software size metrics — quantitative units used to measure the size of software.

SPQR/20 — an automated software size and cost estimation model [JONE86] (see 3.2.2 B i a).

SRR — Software requirements review. Takes place at the end of the feasibility study and is followed by preliminary design.

SSA — Software Sizing Analyser (see 3.2.2 B ii b).

SSM — Software Sizing Model [BOZO86] (see (3.2.2 A i b).

SVAR — a count of the unique variables identifiable at the specification stage from a DBMS sub-schema (see 3.2.2 A ii a).

system adjustment factors — adjustment factors, technical complexity correction factors (see 3.2.2).

system states — number of modes of operation of a software system i.e. training mode, test and mission modes. Used as an input into Price SZ (see 3.2.1 iii).

technical complexity factor — adjustments taking into account the various technical and other factors involved in developing and implementing information processing requirements [ALBR79]. This factor is used to convert raw function points into adjusted function points in FPA and is made up of 14 general information processing adjustments.

technology productivity — a measure of the effect of different development methodologies, tools or environments on software production (see 1.2.1).

tokens — the basic syntactical units distinguishable by a compiler (see 2.2.3).

unadjusted function points see raw function points.

vector metrics — a set of metrics that frame the purpose of the measurement [BASI88] (see 2.4).

V — volume, a Software Science metric, the volume of an implementation of an algorithm in bits (see 2.5.4).

VAR — number of conceptually unique variables (see 3.2.2 A ii a).

variable vectors — set of vectors of variables that characterize a component partition.

vector of candidate variables see 4.3.2.

vocabulary — used in Software Science where the vocabulary of a program is $n = n_1 + n_2$ ie. the sum of unique operators and operands used in the program (see 2.5.4).

volume — a Software Science metric. The volume of an algorithms in bits ($V = N \log_2 n$) (see 2.5.4).

APPENDICES

APPENDIX A

ALL REFERENCE DATA

A1 ALL Reference MMenus

NAME	CHOICE	LOC	Function Points
ADDIM	7	22	3
ADDROM	4	15	3
DATAENT1M	3	14	3
JAJ1	3	14	3
JHMENU	5	18	3
LBLMENU	14	36	3
MICHAEL	13	34	3
MICHAEL2	3	14	3
pend1m	7	22	3
pend1m	6	20	3
PMDENTC	3	14	3
PTADMIN1	10	29	3

A2 ALL Reference SCRMENUS

Name	CHOICE	LOC	Function Points
CMDENTB	3	34	3
CMFENTB	5	42	3
CMFENTB	7	50	3
CMLOOK	9	60	3
CMLOOKT	6	52	3
DARREL	7	53	3
DMDOR	8	52	3
DMOPS	9	57	3
MMDENTA	3	47	3
MMDENTB	16	84	3
mmdentb	16	85	3
mmfenta	4	38	3
PMDENTA	3	34	3
PMDENTB	16	86	3
PMDENTB	16	87	3
PMFENTA	4	37	3
PMFENTB	5	42	3
PMSENTA	4	38	3
SMDENTA	3	34	3
SMDENTB	12	72	3
SMDENTB	12	87	3
SMDENTB	13	75	3
SMDISPAT	5	43	3
SMFENTA	4	38	3
SMSENTA	4	38	3

A3 ALL Reference Relations

Name	Data elements	LOC	Adjusted LOC
Class	2	4	4
COURSE	8	18	9
CRSLBLS	7	12	8
BATCHSTUD	3	5	4
BATCHADDR	15	24	16
PtPendCntrl	2	6	3
PtPendStud	13	28	14
OTHSCHCLASS	3	7	4
EXTEX	2	6	3
FORMGROUP	6	13	7
WDLREASON	2	6	3
WORKFILE	2	4	3
KEYS	2	6	3
LASRNAME	2	5	3
STUDCRS	29	47	31
REPORTLOG	6	8	7
STDCRSSET	14	26	15
SSDUPLICATE	4	7	5
SET	5	12	6
SENREASON	2	6	3
SECSCHOOL	13	27	14
RJWTELECH	3	8	4
StPrePh*	6	15	8
DeptSbj	2	7	4
STUDENT	81	164	82
CAT	2	6	3
ADDREASON	2	4	3
SBJCRSID	2	7	4
STPREXSYR	3	9	4
STPREEXMSUBJ	4	13	5
PRETECH	5	13	6
LOGFILE	2	4	3
LKEYS	2	6	4
FMGRPSTIDS	2	7	3
DEPARTMENT	4	8	5
STPREESBJ	2	7	3
STUDADDR	2	5	3
STEXEX	2	6	3
STEXMSUB	3	8	5
CSFORM	2	4	3
XWDLRSN	2	5	3
ADDRESS	12	23	13
ADDLABELS	33	47	34
TEACHER	8	17	9
STQUAL	2	6	4
TELEXCH	3	8	4
XSTLASTNAME	2	6	3
STPROG	2	6	3
STPREPROG	2	6	3
SUBJECT	4	9	5
STUDLOGS	3	7	4
STPREADDR	3	8	4
STFORM	4	7	6

Name	Data elements	LOC	Adjusted LOC
STCNTRL	2	6	3
SECSCHSTUD	2	7	3
SDTPREEXMDT	5	12	6
STCOMMENT	6	10	7
STCRSSET	2	7	4
STPREEXM	3	7	4

Name	Function Points
Class	7
COURSE	7
CRSLBLS	7
BATCHSTUD	7
BATCHADDR	7
PtPendCntrl	7
PtPendStud	7
OTHSCHCLASS	7
EXTEX	7
FORMGROUP	7
WDLREASON	7
WORKFILE	7
KEYS	7
LASRNAME	7
STUDCRS	7
REPORTLOG	7
STDCRSSET	7
SSDUPLICATE	7
SET	7
SENREASON	7
SECSCHOOL	7
RJWTELECH	7
StPrePh#	7
DeptSbj	7
STUDENT	15
CAT	7
ADDREASON	7
SBJCRSID	7
STPREXSYR	7
STPREEXMSUBJ	7
PRETECH	7
LOGFILE	7
LKEYS	7
FMGRPSTIDS	7
DEPARTMENT	7
STPREESBJ	7
STUDADDR	7
STEXEX	7
STEXMSUB	7
CSFORM	7
XWDLRSN	7
ADDRESS	7
ADDLABELS	7
TEACHER	7
STQUAL	7
TELEXCH	7
XSTLASTNAME	7
STPROG	7
STPREPROG	7
SUBJECT	7
STUDLOGS	7
STPREADDR	7
STFORM	7

Name	Function Points
STCNTRL	7
SECSCHSTUD	7
SDTPREEXMDT	7
STCOMMENT	7
STCRSSET	7
STPREEXM	7

A4 ALL Reference Screens

Name	Logical Scrn	Data elements	CHOICE
Addback	1	16	0
ADDLOOK	1	15	0
ADDMAINT	2	13	2
ADDRSN	1	3	0
AJFIX	1	8	0
AJFIXADD	1	3	0
AJPTCHK	1	11	0
AJSTEXSB	1	15	0
AJWKMK	2	14	0
AJWORKMK	2	16	3
BATCHRCV	1	5	0
CATM	1	3	0
CRSMAINT	1	11	0
CRSWDL	2	10	0
CSCHAWST	1	15	0
CSCHCRAD	1	4	0
CSCHENR	1	8	2
CSCHSET	1	15	0
CSCHSTMK	1	12	0
CSCHSTWD	1	13	0
CSCHSTWD	1	13	0
CSCHWKWK	2	14	2
CSCHWKRN	2	13	2
CSCHWKRV	2	16	2
CSCHXMEN	2	19	5
CSCRSAD1	2	12	2
CSCRSAD1	2	12	2
CSCRSAD2	1	9	3
CSCRSWD1	2	15	2
CSCRSWD2	2	14	4
CSCSTADR	2	12	3
CSCSTNME	1	16	3
CSCSTPH	2	16	3
CSDSTCR	1	13	0
CSEXSN	2	18	2
CSFMSENT	2	11	2
CSFORMM	1	3	0
CSFMSENTX	2	11	2
CSLCAW	3	23	2
CSLCLSRL	2	12	2
CSLENR	1	3	2
CSLINSRL	2	10	2
CSLKENR	1	8	2
CSLKFMLST	2	10	2
CSLKFMLSTX	3	10	2
CSLKSTCM	2	12	2
CSLKSTCN	2	12	2
CSLKSUM	3	25	2
CSLKSUM	3	25	2
CSLKSUMX	3	21	2
CSLKSUMX	3	21	2
CSLPREAP	3	22	2
CSLCSHRL	2	10	2

Name	Logical Scrn	Data elements	CHOICE
CSLSCHRL	2	10	2
CSLSRNM1	2	9	2
CSLSRNME	2	9	2
CSLWORK	3	7	3
CSLWORK1	3	16	3
CSLWORK1X	3	16	3
CSLXMENT	2	15	2
CSQLADSB	1	18	2
CSQLENT	2	25	2
CSQLLOOK	2	24	2
CSQLWDEN	2	23	2
CSQLWDSB	1	18	2
CSSTCOM	2	12	2
CSSTDEF	2	18	2
CSSTUNWD	1	13	0
CSSTWD1	2	10	2
CSSTWD2	2	20	0
CSSSTWDX	2	11	2
CSWKMRK	2	14	2
EXMSUBMT	1	7	0
EXMCODEM	1	5	0
FGMAINT	1	6	0
FSBENO1	1	9	0
FSBEND2	1	8	0
FTQKCHNG	2	24	2
INSTMAIN	1	10	0
IPSTCHEN	1	6	0
IXMSCALE	1	6	0
JAJCP3M5	2	10	3
JANESFIX	1	14	0
JHCRSWDL	2	10	0
JJBACK	2	18	4
JJCP3M5	2	13	3
JJTEST	1	19	4
JJTEST1	2	19	4
LBLADD	1	13	2
LBLCHNGE	2	13	2
LBLLOOK	2	13	2
LBLOSAD	1	17	2
LBLREQST	1	9	2
MORELBLS	1	8	3
MORELOGS	1	6	3
MSCHENR1	1	28	4
MSCHENR1	1	28	4
MSCHENR2	2	13	4
MSCHENR3	4	12	4
MSCHENR4	3	23	3
MSCHENR4	3	23	3
MSENR1	1	27	2
MSENR2	2	13	2
MSENR3	2	10	0
MSENR4	2	23	3
MSLKENR1	1	27	5

Name	Logical Scrn	Data elements	CHOICE
MSLKENR2	2	13	4
MSLKENR3	4	12	4
MSLKENR4	3	23	2
MWFIXAJ	1	6	0
PEND1	1	12	3
PRND2	1	14	3
PEND2	1	14	3
PEND3	1	14	3
PEND4	1	14	3
PENDR2	1	14	0
PSCENR2	2	10	0
PSCHENR	1	19	0
PSCHENR	1	19	0
PSCHENR1	2	13	4
PSCHENR2	4	12	4
PSCHENR3	3	23	3
PSCNENR1	2	13	2
PSNENR2	2	10	0
PSCNENR3	2	23	3
PSCRSAD1	2	12	2
PSCRSAD2	1	9	3
PSINENR	1	20	3
PSLKENR	2	20	5
PSLKENR1	2	13	4
PSLKENR2	4	12	4
PSLKENR3	3	23	2
PSWKRCV	2	13	3
SBJMAINT	2	7	0
SCHLMAIN	1	9	3
SCHLMAIN	1	9	3
SCHRPT	1	4	2
SETMAIN	1	7	0
SETMAINT	1	6	0
SSCH1	2	17	4
SSCH2	3	13	3
SSCH3	3	15	2
SSCHENR1	2	19	4
SSCHENR1	2	19	4
SSCHENR2	3	15	3
SSCHENR2	3	13	3
SSCHENR3	5	16	2
SSCHENR3	5	16	2
SSCRSAD1	2	12	2
SSCRSAD2	1	11	3
SSDUPL	1	7	2
SSENR1	2	18	1
SSENR2	3	13	1
SSENR3	3	15	2
SSENROL	2	17	1
SSLBLS	1	5	0
SSLKENR1	2	17	4
SSLKENR2	3	13	3
SSLKENR3	5	15	3

Name	Logical Scrn	Data elements	CHOICE
SSWKRCV	2	16	2
SSWKRTN	2	13	2
TCHMAINT	1	5	0
WKRECCHG	1	15	1
XCXLWORK	2	17	1
XSSCHENR	2	19	4

Name	Function Points	Logic	Relations
Addback	6	44	3
ADDLOOK	4	0	1
ADDMAINT	7	89	2
ADDRSN	3	0	1
AJFIX	4	9	2
AJFIXADD	3	0	1
AJPTCHK	4	0	1
AJSTEXSB	4	0	1
AJWKMK	10	0	7
AJWORKMK	9	179	10
BATCHRCV	3	0	7
CATM	3	0	7
CRSMAINT	3	0	1
CRSWDL	7	10	4
CSCHAWST	3	2	1
CSCHCRAD	3	0	1
CSCHENR	8	37	2
CSCHSET	3	2	1
CSCHSTMK	7	66	6
CSCHSTWD	3	0	1
CSCHSTWD	3	0	3
CSCHWKWK	9	26	6
CSCHWKRN	10	25	7
CSCHWKRV	10	31	7
CSCHXMEN	10	74	6
CSCRSAD1	10	10	4
CSCRSAD1	10	10	4
CSCRSAD2	9	41	4
CSCRSWD1	10	26	4
CSCRSWD2	10	31	4
CSCSTADR	7	21	2
CSCSTNME	8	38	3
CSCSTPH	8	39	3
CSDSTCR	3	0	1
CSEXSN	9	56	5
CSFMSENT	9	28	4
CSFORMM	3	0	1
CSFMSENTX	9	28	4
CSLCAW	7	24	8
CSLCLSRL	10	13	5
CSLENR	6	15	1
CSLINSRL	8	11	3
CSLKENR	8	30	2
CSLKFMLST	8	20	3
CSLKFMLSTX	8	20	3
CSLKSTCM	8	45	2
CSLKSTCN	8	45	2
CSLKSUM	10	34	6
CSLKSUM	10	34	6
CSLKSUMX	10	42	7
CSLKSUMX	10	36	7
CSLPREAP	10	13	3
CSLCSHRL	8	9	3

Name	Function Points	Logic	Relations
CSLSCHRL	8	9	3
CSLSRNM1	8	14	3
CSLSRNME	8	15	3
CSLWORK	8	17	3
CSLWORK1	10	21	4
CSLWORK1X	10	21	4
CSLXMENT	10	17	5
CSQLADSB	10	55	7
CSQLENT	10	128	11
CSQLLOOK	10	51	7
CSQLWDEN	10	75	7
CSQLWDSB	8	41	3
CSSTCOM	9	28	3
CSSTDEF	9	65	3
CSSTUNWD	4	0	2
CSSTWD1	8	13	2
CSSTWD2	7	38	6
CSSSTWDX	8	18	2
CSWKMRK	9	27	6
EXMSUBMT	3	0	1
EXMCODEM	3	0	1
FGMAINT	3	0	1
FSBENO1	6	95	4
FSBEND2	6	24	4
FTQKCHNG	9	125	9
INSTMAIN	3	0	1
IPSTCHEN	4	3	3
IXMSCALE	3	0	1
JAJCP3M5	10	15	6
JANESFIX	3	0	1
JHCRSWDL	7	4	4
JJBACK	9	42	7
JJCP3M5	9	21	5
JJTEST	9	42	7
JJTEST1	9	42	3
LBLADD	6	71	1
LBLCHNGE	6	96	1
LBLLOOK	7	91	1
LBLOSAD	6	70	1
LBLREQST	7	32	1
MORELBLS	7	27	2
MORELOGS	8	25	4
MSCHENR1	9	96	7
MSCHENR1	9	197	7
MSCHENR2	7	44	2
MSCHENR3	9	19	6
MSCHENR4	9	33	3
MSCHENR4	9	33	4
MSENR1	9	210	9
MSENR2	7	39	2
MSENR3	6	4	6
MSENR4	9	35	6
MSLKENR1	10	176	3

Name	Function Points	Logic	Relations
MSLKENR2	8	16	2
MSLKENR3	10	16	6
MSLKENR4	10	12	4
MWFIXAJ	3	0	1
PEND1	6	12	1
PRND2	7	11	1
PEND2	7	11	1
PEND3	7	10	1
PEND4	6	13	1
PENDR2	3	0	1
PSCENR2	6	4	6
PSCHENR	6	65	5
PSCHENR	6	63	5
PSCHENR1	7	43	2
PSCHENR2	9	19	6
PSCHENR3	9	23	4
PSCNENR1	7	37	2
PSNENR2	6	4	6
PSCNENR3	9	35	5
PSCRSAD1	10	10	4
PSCRSAD2	9	28	6
PSINENR	9	87	6
PSLKENR	10	29	3
PSLKENR1	8	16	2
PSLKENR2	10	16	6
PSLKENR3	10	12	3
PSWKRCV	10	41	6
SBJMAINT	6	0	4
SCHLMAIN	7	7	1
SCHLMAIN	7	6	1
SCHRPT	7	15	1
SETMAIN	6	5	3
SETMAINT	6	7	3
SSCH1	9	41	7
SSCH2	9	29	4
SSCH3	9	24	7
SSCHENR1	9	47	3
SSCHENR1	9	55	3
SSCHENR2	9	33	5
SSCHENR2	9	18	4
SSCHENR3	10	30	6
SSCHENR3	10	29	7
SSCRSAD1	10	10	4
SSCRSAD2	9	29	7
SSDUPL	7	30	2
SSENR1	10	62	8
SSENR2	9	18	5
SSENR3	9	36	7
SSENROL	9	43	7
SSLBLS	4	2	1
SSLKENR1	10	19	5
SSLKENR2	10	13	4
SSLKENR3	10	15	7

Name	Function Points	Logic	Relations
SSWKRCV	10	38	7
SSWKRTN	10	25	7
TCHMAINT	3	0	1
WKRECCHG	9	8	5
XCXLWORK	10	11	5
XSSCHENR	9	42	7

Name	cplx3	cplx5	LOC
Addback	38	41	90
ADDLOOK	0	0	43
ADDMAINT	38	95	138
ADDRSN	0	0	22
AJFIX	0	11	41
AJFIXADD	0	0	22
AJPTCHK	0	0	37
AJSTEXSB	0	0	46
AJWKMK	0	0	51
AJWORKMK	38	95	242
BATCHRCV	0	0	26
CATM	0	0	22
CRSMAINT	0	0	36
CRSWDL	0	11	55
CSCHAWST	0	0	50
CSCHCRAD	0	0	23
CSCHENR	38	41	65
CSCHSET	0	0	49
CSCHSTMK	38	95	109
CSCHSTWD	0	0	43
CSCHSTWD	0	0	45
CSCHWKWK	17	23	76
CSCHWKRN	17	23	75
CSCHWKRV	38	41	84
CSCHXMEN	38	95	133
CSCRSAD1	0	11	59
CSCRSAD1	0	11	57
CSCRSAD2	38	41	77
CSCRSWD1	17	23	77
CSCRSWD2	38	41	80
CSCSTADR	17	23	65
CSCSTNME	38	41	81
CSCSTPH	38	41	89
CSDSTCR	0	0	41
CSEXSN	38	41	106
CSFMSENT	17	23	72
CSFORMM	0	0	21
CSFMSENTX	17	23	72
CSLCAW	17	23	98
CSLCLSRL	17	11	61
CSLENR	17	11	37
CSLINSRL	0	11	55
CSLKENR	38	23	58
CSLKFMLST	17	23	63
CSLKFMLSTX	17	23	62
CSLKSTCM	38	41	88
CSLKSTCN	38	41	88
CSLKSUM	38	41	102
CSLKSUM	38	41	102
CSLKSUMX	38	41	107
CSLKSUMX	38	41	101
CSLPREAP	17	11	80
CSLCSHRL	0	11	53

Name	cplx3	cplx5	LOC
CSLSCHRL	0	11	53
CSLSRNM1	17	11	54
CSLSRNME	17	11	55
CSLWORK	17	23	63
CSLWORK1	17	23	78
CSLWORK1X	17	23	78
CSLXMENT	17	23	70
CSQLADSB	38	41	107
CSQLENT	38	95	198
CSQLLOOK	38	41	118
CSQLWDEN	38	95	142
CSQLWDSB	38	41	88
CSSTCOM	17	23	72
CSSTDEF	38	95	112
CSSTUNWD	0	0	45
CSSTWD1	17	11	52
CSSTWD2	17	41	96
CSSSTWDX	17	23	66
CSWKMRK	17	23	77
EXMSUBMT	0	0	29
EXMCODEM	0	0	25
FGMAINT	0	0	28
FSBENO1	38	95	133
FSBEND2	17	23	60
FTQKCHNG	38	95	195
INSTMAIN	0	0	36
IPSTCHEN	0	0	32
IXMSCALE	0	0	26
JAJCP3M5	17	11	59
JANESFIX	0	0	44
JHCRSWDL	0	0	48
JJBACK	38	41	102
JJCP3M5	17	23	68
JJTEST	38	41	99
JJTEST1	38	41	97
LBLADD	38	95	114
LBLCHNGE	38	95	142
LBLLOOK	38	95	138
LBLOSAD	38	95	119
LBLREQST	38	41	70
MORELBLS	17	23	57
MORELOGS	17	23	56
MSCHENR1	38	95	165
MSCHENR1	38	95	266
MSCHENR2	38	41	89
MSCHENR3	17	23	84
MSCHENR4	38	41	101
MSCHENR4	38	41	101
MSENR1	38	95	280
MSENR2	38	41	84
MSENR3	0	0	48
MSENR4	38	41	97
MSLKENR1	38	95	237

Name	cplx3	cplx5	LOC
MSLKENR2	17	23	60
MSLKENR3	17	23	79
MSLKENR4	17	11	78
MWFIXAJ	0	0	27
PEND1	17	11	53
PRND2	0	11	56
PEND2	0	11	56
PEND3	0	11	56
PEND4	17	11	59
PENDR2	0	0	43
PSCENR2	0	0	48
PSCHENR	38	95	123
PSCHENR	38	95	121
PSCHENR1	38	41	88
PSCHENR2	17	23	84
PSCHENR3	17	23	91
PSCNENR1	38	41	82
PSNENR2	0	0	49
PSCNENR3	38	41	96
PSCRSAD1	0	11	57
PSCRSAD2	17	23	65
PSINENR	38	95	147
PSLKENR	17	23	88
PSLKENR1	17	23	60
PSLKENR2	17	23	79
PSLKENR3	17	11	80
PSWKRCV	38	41	90
SBJMAINT	0	0	37
SCHLMAIN	0	11	38
SCHLMAIN	0	11	38
SCHRPT	17	11	44
SETMAIN	0	0	35
SETMAINT	0	11	36
SSCH1	38	41	100
SSCH2	17	23	85
SSCH3	17	23	85
SSCHENR1	38	41	101
SSCHENR1	38	41	108
SSCHENR2	38	41	92
SSCHENR2	17	23	74
SSCHENR3	38	23	134
SSCHENR3	17	23	135
SSCRSAD1	0	11	57
SSCRSAD2	17	23	71
SSDUPL	38	23	66
SSENR1	38	95	124
SSENR2	17	23	74
SSENR3	38	41	94
SSENROL	38	41	102
SSLBLS	0	0	26
SSLKENR1	17	23	73
SSLKENR2	17	11	68
SSLKENR3	17	11	114

Name	cplx3	cplx5	LOC
SSWKRCV	38	41	92
SSWKRTN	17	23	78
TCHMAINT	0	0	26
WKRECCHG	0	11	52
XCXLWORK	0	11	63
XSSCHENR	38	41	104

Name	Tokens	LOC1	LINES
Addback	121	46	90
ADDLOOK	90	43	43
ADDMAINT	116	49	50
ADDRSN	36	22	22
AJFIX	74	32	32
AJFIXADD	31	22	22
AJPTCHK	68	37	37
AJSTEXSB	86	46	46
AJWKMK	165	51	56
AJWORKMK	232	63	71
BATCHRCV	47	26	26
CATM	36	22	22
CRSMAINT	73	36	36
CRSWDL	120	45	52
CSCHAWST	108	48	48
CSCHCRAD	43	23	23
CSCHENR	82	28	29
CSCHSET	108	47	47
CSCHSTMK	147	43	44
CSCHSTWD	97	43	43
CSCHSTWD	109	45	45
CSCHWKWK	149	50	53
CSCHWKRN	150	50	53
CSCHWKRV	164	53	56
CSCHXMEN	188	59	66
CSCRSAD1	132	49	57
CSCRSAD1	129	47	57
CSCRSAD2	101	36	38
CSCRSWD1	155	51	59
CSCRSWD2	138	49	54
CSCSTADR	109	44	46
CSCSTNME	124	43	44
CSCSTPH	143	50	52
CSDSTCR	95	41	42
CSEXSN	163	50	53
CSFMSENT	131	44	47
CSFORMM	36	21	21
CSFMSENTX	125	44	47
CSLCAW	252	74	88
CSLCLSRL	136	48	56
CSLENR	47	22	23
CSLINSRL	114	44	52
CSLKENR	82	28	29
CSLKFMLST	113	43	50
CSLKFMLSTX	112	42	49
CSLKSTCM	114	43	50
CSLKSTCN	114	43	50
CSLKSUM	226	68	81
CSLKSUM	169	68	81
CSLKSUMX	221	65	78
CSLKSUMX	221	65	78
CSLPREAP	178	67	79
CSLCSHRL	124	44	51

Name	Tokens	LOC1	LINES
CSLSCHRL	114	44	51
CSLSRNM1	100	40	48
CSLSRNME	100	40	48
CSLWORK	99	46	51
CSLWORK1	166	57	70
CSLWORK1X	167	57	71
CSLXMENT	157	53	59
CSQLADSB	174	52	54
CSQLENT	255	70	76
CSQLLOOK	220	67	74
CSQLWDEN	227	67	74
CSQLWDSB	140	47	49
CSSTCOM	122	44	47
CSSTDEF	148	47	50
CSSTUNWD	103	45	45
CSSTWD1	102	39	45
CSSTWD2	186	58	65
CSSSTWDX	114	48	51
CSWKMRK	154	50	53
EXMSUBMT	53	29	29
EXMCODEM	46	25	25
FGMAINT	50	28	28
FSBENO1	115	38	39
FSBEND2	111	36	38
FTQKCHNG	213	70	77
INSTMAIN	62	36	36
IPSTCHEN	78	29	29
IXMSCALE	51	26	27
JAJCP3M5	121	44	46
JANESFIX	83	44	44
JHCRSWDL	110	44	50
JJBACK	199	60	62
JJCP3M5	132	47	48
JJTEST	211	57	58
JJTEST1	161	55	57
LBLADD	98	43	44
LBLCHNGE	100	46	48
LBLLOOK	114	47	49
LBLOSAD	109	49	50
LBLREQST	83	38	39
MORELBLS	79	30	31
MORELOGS	84	31	32
MSCHENR1	226	69	70
MSCHENR1	229	69	71
MSCHENR2	113	45	47
MSCHENR3	180	65	83
MSCHENR4	186	68	80
MSCHENR4	186	68	80
MSENR1	248	70	71
MSENR2	112	45	48
MSENR3	138	44	50
MSENR4	202	62	70
MSLKENR1	183	61	63

Name	Tokens	LOC1	LINES
MSLKENR2	114	44	47
MSLKENR3	163	63	78
MSLKENR4	192	66	79
MWFIXAJ	47	27	27
PEND1	89	41	43
PRND2	104	45	46
PEND2	103	45	45
PEND3	101	46	46
PEND4	92	46	47
PENDR2	81	43	43
PSCENR2	137	44	50
PSCHENR	172	58	59
PSCHENR	170	58	58
PSCHENR1	113	45	47
PSCHENR2	185	65	83
PSCHENR3	187	68	80
PSCNENR1	113	45	47
PSNENR2	128	45	51
PSCNENR3	196	61	69
PSCRSAD1	131	47	55
PSCRSAD2	114	37	39
PSINENR	187	60	61
PSLKENR	162	59	61
PSLKENR1	117	44	47
PSLKENR2	165	63	78
PSLKENR3	194	68	80
PSWKRCV	146	49	52
SBJMAINT	93	37	42
SCHLMAIN	64	31	31
SCHLMAIN	64	32	32
SCHRPT	57	29	30
SETMAIN	75	30	30
SETMAINT	69	29	29
SSCH1	195	59	61
SSCH2	154	56	68
SSCH3	173	61	74
SSCHENR1	155	54	56
SSCHENR1	157	53	56
SSCHENR2	163	59	71
SSCHENR2	142	56	68
SSCHENR3	218	104	102
SSCHENR3	225	106	104
SSCRSAD1	130	47	55
SSCRSAD2	133	42	44
SSDUPL	88	36	37
SSENR1	220	62	64
SSENR2	151	56	65
SSENR3	168	58	68
SSENROL	201	59	61
SSLBLS	47	24	24
SSLKENR1	153	54	56
SSLKENR2	144	55	66
SSLKENR3	217	99	97

Name	Tokens	LOC1	LINES
SSWKRCV	169	54	56
SSWKRTN	169	53	53
TCHMAINT	42	26	26
WKRECCHG	124	44	44
XCXLWORK	160	52	59
XSSCHENR	215	62	64

A5 ALL Reference Reports

Name	Data elements	Line types	LOC
ADLBLCR	3	2	78
ADLBLCR	3	2	77
ADFTSFC	28	2	125
ADFTSFC	13	11	127
ADFTSFC	30	5	134
ADFTSFC	13	6	125
CATREP	3	1	18
CHECKBT	15	1	46
CLASSCH	8	1	32
CLASSLS	7	2	26
COD	4	1	23
COD3	19	17	119
COMTRPT	7	7	50
COUNTGU	11	9	50
CRSHQWD	6	3	26
CRCHQWD	11	0	26
CRDWD	14	8	66
CRNOFMG	5	1	22
CRSCH	3	1	28
CRSLIST	12	8	49
CRSLIST	12	8	111
CRSLIST	12	8	47
CRSLTES	12	8	107
CRSMKFI	4	2	83
CRSNME	3	1	17
CRSUBJ	12	8	50
CRSWDLH	14	8	61
CRSWDLH	14	8	66
CRZIP	3	1	23
CS/SS/5	18	9	60
CSSS80F	15	13	110
CSSS80	15	10	112
DATACHK	24	4	59
DATACHK	24	4	58
DATACHK	14	4	114
DEANZ	4	1	26
DEFAULT	16	9	70
DEFAULT	16	9	68
DEFAULT	16	8	63
DJCHECK	14	10	70
DJCHECK	21	9	99
DJFMLST	17	8	97
DJINSLS	7	5	35
FORMLIST	9	3	31
FORMLIST	10	3	31
FPLOGCT	15	17	129
FTSSG	10	3	35
FTSSG	10	3	35
IXMRANK	16	10	69
IXMRANK	15	10	70
IXMST11	55	65	432
IXMST12	55	65	433
IXMSTAA	55	65	432

Name	Data elements	Line types	LOC
IXMSTAB	55	65	433
IXMSTAT	55	65	432
IXMSTAT	55	65	432
IXMSTA	55	65	431
LBLIST	8	2	24
LBLLIST	8	2	24
LBLNZCH	11	2	34
LOGCT1	8	5	39
LOGCT1	12	5	56
LOGCT2	7	5	37
LOGCT2	7	5	38
LOGCT3	9	5	41
LOGCT4	20	8	62
LOGCTJH	14	17	130
LOGHDR1	34	10	82
LOGHDR5	27	0	112
MSCRSLS	12	10	51
MSFMLST	20	8	102
MWLOGHD	32	1	126
NMEDIST	7	4	33
PENDR3	17	15	54
PENDRPT	7	4	25
PENDRPT	7	4	24
PROLLCT	12	8	45
PROLLCT	11	8	45
PROLLCT	11	7	51
PROLLCT	11	8	58
PROLLCT	13	7	62
PROLLCT	13	7	63
PRFCMAY	15	15	123
PTFCMAY	40	28	156
PTFMLST	11	10	68
PRLOGHD	28	1	120
PTMAYRO	11	16	76
PTNO2DE	6	3	30
REPORT1	6	8	51
ROLLCT1	11	4	38
SCHLILST	4	3	18
SCHROL1	6	4	51
SCHROL2	19	2	72
SCHROL	6	2	30
SCHROL	14	3	70
SCHROLL	15	3	64
SCHRPT1	14	2	63
SCHRPTA	16	2	66
SCHTOLD	8	4	39
SCHRPT	12	10	34
SETDELA	19	9	75
SETDELAY	19	9	79
SFCRANK	3	2	45
SFCRANK	3	2	52
SFCRANK	3	2	51
SFCSTAT	9	9	239

Name	Data elements	Line types	LOC
SILOGHD	33	2	154
SMFMLST	13	10	77
SROLLCT	11	8	48
SROLCT1	11	8	45
SSACMAY	15	15	123
SSACMAY	41	29	239
SSACTMAY	40	29	285
SSCHK1	18	4	133
SSCHK1	18	4	135
SSCHK2	12	7	68
SSCHK3	12	11	77
SSCHK3	13	11	77
SSCHKAJ	11	9	74
SSFMLST	16	8	75
SSFMLST	20	10	105
SSFTMAY	15	15	123
SSFTMAY	40	28	240
SSGFORM	8	5	32
SSGRPT1	24	26	137
SSGRPT2	39	28	205
SSLOGHD	33	2	139
SSLOGHD	29	2	128
SSLOGHD	30	2	129
SSLOGRP	30	2	127
SSMAYRO	40	28	258
TEST	6	1	20

Name	Logic	Relations	Logical reports
IXMSTAB	292	7	1
IXMSTAT	291	7	1
IXMSTAT	291	7	1
IXMSTA	290	7	1
LBLIST	1	1	1
LBLLIST	1	1	1
LBLNZCH	8	1	1
LOGCT1	5	3	1
LOGCT1	16	4	1
LOGCT2	5	2	1
LOGCT2	6	2	1
LOGCT3	5	4	1
LOGCT4	12	4	1
LOGCTJH	51	5	1
LOGHDR1	21	6	1
LOGHDR5	58	8	3
MSCRSLS	6	6	1
MSFMLST	43	5	1
MWLOGHD	63	10	3
NMEDIST	7	1	1
PENDR3	7	1	1
PENDRPT	2	1	1
PENDRPT	1	1	1
PROLLCT	7	2	1
PROLLCT	7	2	1
PROLLCT	16	1	1
PROLLCT	21	1	1
PROLLCT	24	1	1
PROLLCT	24	1	1
PRFCMAY	46	2	1
PTFCMAY	86	2	1
PTFMLST	22	4	2
PRLOGHD	62	9	3
PTMAYRO	33	2	1
PTNO2DE	7	1	1
REPORT1	18	1	1
ROLLCT1	7	2	1
SCHLILST	0	1	1
SCHROL1	20	4	2
SCHROL2	21	5	3
SCHROL	2	3	2
SCHROL	21	5	3
SCHROLL	15	4	3
SCHRPT1	21	5	3
SCHRPTA	21	5	3
SCHTOLD	5	4	2
SCHRPT	0	1	1
SETDELA	13	5	1
SETDELAY	16	5	1
SFCRANK	21	6	1
SFCRANK	28	6	1
SFCRANK	27	6	1
SFCSTAT	201	7	1

Name	Logic	Relations	Logical reports
SILOGHD	82	12	4
SMFMLST	25	5	2
SROLLCT	10	2	1
SROLCT1	7	2	1
SSACMAY	46	2	1
SSACMAY	109	3	1
SSACTMAY	86	2	1
SSCHK1	79	10	4
SSCHK1	79	10	4
SSCHK2	22	4	2
SSCHK3	25	5	2
SSCHK3	25	5	2
SSCHKAJ	28	4	2
SSFMLST	20	5	2
SSFMLST	43	5	2
SSFTMAY	46	2	1
SSFTMAY	86	2	1
SSGFORM	3	2	1
SSGRPT1	74	1	1
SSGRPT2	111	1	1
SSLOGHD	70	11	4
SSLOGHD	68	10	3
SSLOGHD	68	10	3
SSLOGRP	67	9	3
SSMAYRO	88	2	1
TEST	0	1	1

Name	Logic	Relations	Logical reports
ADLBLCR	53	6	1
ADLBLCR	52	6	1
ADFTSFC	71	8	2
ADFTSFC	82	7	1
ADFTSFC	73	9	2
ADFTSFC	82	7	1
CATREP	1	1	1
CHECKBT	14	3	1
CLASSCH	3	4	2
CLASSLS	0	3	2
COD	3	2	1
COD3	37	2	1
COMTRPT	10	2	1
COUNTGU	12	2	1
CRSHQWD	4	2	1
CRCHQWD	4	2	1
CRDWD	19	7	1
CRNOFMG	3	1	1
CRSCH	9	3	1
CRSLIST	6	6	1
CRSLIST	65	9	1
CRSLIST	4	6	1
CRSLTES	61	9	1
CRSMKFI	61	2	1
CRSNME	0	2	1
CRSUBJ	7	6	1
CRSWDLH	14	7	1
CRSWDLH	19	7	1
CRZIP	5	2	1
CS/SS/5	7	4	3
CSSS80F	58	5	1
CSSS80	63	5	1
DATACHK	16	3	1
DATACHK	15	3	1
DATACHK	65	6	4
DEANZ	7	2	1
DEFAULT	18	4	1
DEFAULT	16	4	1
DEFAULT	13	4	1
DJCHECK	20	5	1
DJCHECK	31	7	3
DJFMLST	43	5	2
DJINSLS	6	2	1
FORMLIST	3	2	1
FORMLIST	3	2	1
FPLOGCT	49	5	1
FTSSG	7	2	1
FTSSG	7	2	1
IXMRANK	17	5	1
IXMRANK	17	5	1
IXMST11	291	7	1
IXMST12	292	7	1
IXMSTAA	291	7	1

Name	Sort fields	Control breaks	cplx3
ADLBLCR	3	2	70
ADLBLCR	3	2	70
ADFTSFC	2	0	70
ADFTSFC	4	2	70
ADFTSFC	3	1	70
ADFTSFC	4	2	70
CATREP	1	0	5
CHECKBT	1	0	20
CLASSCH	3	0	5
CLASSLS	1	0	5
COD	1	1	5
COD3	2	2	20
COMTRPT	1	2	5
COUNTGU	2	3	20
CRSHQWD	0	0	5
CRCHQWD	0	0	5
CRDWD	5	3	20
CRNOFMG	1	0	5
CRSCH	2	0	5
CRSLIST	5	2	5
CRSLIST	5	2	70
CRSLIST	4	2	5
CRSLTES	5	2	70
CRSMKFI	3	2	70
CRSNME	0	0	5
CRSUBJ	5	2	5
CRSWDLH	5	3	20
CRSWDLH	5	3	20
CRZIP	1	0	5
CS/SS/5	4	2	5
CSSS80F	5	2	70
CSSS80	5	2	70
DATACHK	1	0	20
DATACHK	1	0	20
DATACHK	1	0	70
DEANZ	2	0	5
DEFAULT	3	2	20
DEFAULT	3	2	20
DEFAULT	2	1	20
DJCHECK	4	2	20
DJCHECK	7	3	20
DJFMLST	3	2	20
DJINSLS	2	0	5
FORMLIST	2	1	5
FORMLIST	2	1	5
FPLOGCT	3	3	70
FTSSG	2	1	5
FTSSG	2	1	5
IXMRANK	2	1	20
IXMRANK	4	2	20
IXMST11	2	2	70
IXMST12	2	2	70
IXMSTAA	2	2	70

Name	Sort fields	Control breaks	cplx3
IXMSTAB	2	2	70
IXMSTAT	2	2	70
IXMSTAT	2	2	70
IXMSTA	2	2	70
LBLIST	1	0	5
LBLLIST	5	0	5
LBLNZCH	2	0	5
LOGCT1	1	2	5
LOGCT1	1	2	20
LOGCT2	1	2	5
LOGCT2	1	2	5
LOGCT3	1	2	5
LOGCT4	3	4	20
LOGCTJH	2	3	70
LOGHDR1	4	1	20
LOGHDR5	4	0	70
MSCRSLS	5	2	5
MSFMLST	3	2	20
MWLOGHD	5	1	70
NMEDIST	1	1	5
PENDR3	1	0	5
PENDRPT	1	0	5
PENDRPT	1	0	5
PROLLCT	1	2	5
PROLLCT	1	2	5
PROLLCT	1	2	20
PROLLCT	1	2	20
PROLLCT	1	2	20
PROLLCT	1	2	20
PRFCMAY	2	1	70
PTFCMAY	1	2	70
PTFMLST	3	1	20
PRLOGHD	5	2	70
PTMAYRO	2	2	20
PTNO2DE	0	1	5
REPORT1	2	2	20
ROLLCT1	1	2	5
SCHLILST	1	0	5
SCHROL1	2	0	20
SCHROL2	4	0	20
SCHROL	1	0	5
SCHROL	3	0	20
SCHROLL	3	0	20
SCHRPT1	2	0	20
SCHRPTA	2	0	20
SCHTOLD	4	1	5
SCHRPT	0	0	5
SETDELA	5	3	20
SETDELAY	5	3	20
SFCRANK	2	1	20
SFCRANK	2	1	20
SFCRANK	2	1	20
SFCSTAT	1	1	70

Name	Sort fields	Control breaks	cplx3
SILOGHD	6	2	70
SMFMLST	5	2	20
SROLLCT	1	2	5
SROLCT1	1	2	5
SSACMAY	2	1	70
SSACMAY	2	2	70
SSACTMAY	1	2	70
SSCHK1	1	0	70
SSCHK1	1	0	70
SSCHK2	5	2	20
SSCHK3	5	2	20
SSCHK3	5	2	20
SSCHKAJ	2	1	20
SSFMLST	3	1	20
SSFMLST	3	2	20
SSFTMAY	2	1	70
SSFTMAY	1	2	70
SSGFORM	2	1	5
SSGRPT1	0	1	70
SSGRPT2	2	3	70
SSLOGHD	6	1	70
SSLOGHD	5	1	70
SSLOGHD	5	1	70
SSLOGRP	5	1	70
SSMAYRO	1	2	70
TEST	5	0	5

Name	LINES	LOC1	function points	cplx5
ADLBLCR	24	25	5	46
ADLBLCR	24	25	5	46
ADFTSFC	52	54	7	82
ADFTSFC	38	45	7	82
ADFTSFC	57	61	7	82
ADFTSFC	33	43	7	82
CATREP	17	17	4	3
CHECKBT	32	32	5	7
CLASSCH	29	29	7	3
CLASSLS	24	26	5	3
COD	19	20	4	3
COD3	35	82	5	46
COMTRPT	31	40	5	7
COUNTGU	28	38	5	7
CRSHQWD	21	22	5	3
CRCHQWD	18	22	5	3
CRDWD	36	47	7	20
CRNOFMG	18	19	4	3
CRSCH	19	19	4	7
CRSLIST	32	43	7	3
CRSLIST	35	46	7	82
CRSLIST	32	43	7	3
CRSLTES	35	46	7	46
CRSMKFI	21	22	4	46
CRSNME	17	17	4	3
CRSUBJ	32	43	7	7
CRSWDLH	36	47	7	7
CRSWDLH	36	47	7	20
CRZIP	18	18	4	3
CS/SS/5	49	53	7	7
CSSS80F	35	49	7	46
CSSS80	38	52	7	46
DATACHK	40	43	7	20
DATACHK	40	43	7	7
DATACHK	45	49	7	82
DEANZ	19	19	4	7
DEFAULT	35	52	7	20
DEFAULT	35	52	7	20
DEFAULT	34	50	7	7
DJCHECK	37	50	7	20
DJCHECK	55	68	7	46
DJFMLST	40	54	7	46
DJINSLS	23	29	5	3
FORMLIST	22	28	5	3
FORMLIST	22	28	5	3
FPLOGCT	41	80	7	46
FTSSG	22	28	5	7
FTSSG	22	28	5	7
IXMRANK	36	52	7	20
IXMRANK	37	53	7	20
IXMST11	87	164	7	82
IXMST12	87	164	7	82
IXMSTAA	87	164	7	82

Name	LINES	LOC1	function points	cplx5
IXMSTAB	87	164	7	82
IXMSTAT	87	164	7	82
IXMSTAT	87	164	7	82
IXMSTA	87	164	7	82
LBLIST	20	23	4	3
LBLLIST	20	23	4	3
LBLNZCH	24	26	4	7
LOGCT1	21	34	5	3
LOGCT1	27	40	7	20
LOGCT2	24	32	5	3
LOGCT2	24	32	5	3
LOGCT3	28	36	7	3
LOGCT4	35	50	7	7
LOGCTJH	52	90	7	46
LOGHDR1	51	64	7	20
LOGHDR5	54	54	7	46
MSCRSLS	34	45	7	3
MSFMLST	41	59	7	46
MWLOGHD	62	63	7	46
NMEDIST	21	26	4	7
PENDR3	41	47	4	7
PENDRPT	21	23	4	3
PENDRPT	21	23	4	3
PROLLCT	28	38	5	7
PROLLCT	28	38	5	7
PROLLCT	23	35	4	20
PROLLCT	25	37	4	20
PROLLCT	24	38	4	20
PROLLCT	24	39	4	20
PRFCMAY	32	77	5	46
PTFCMAY	168	192	7	82
PTFMLST	36	46	7	20
PRLOGHD	57	58	7	46
PTMAYRO	35	43	5	46
PTNO2DE	19	23	4	7
REPORT1	25	33	4	20
ROLLCT1	24	31	5	7
SCHLILST	19	18	4	3
SCHROL1	28	31	7	20
SCHROL2	47	51	7	20
SCHROL	24	28	5	3
SCHROL	41	49	7	20
SCHROLL	41	49	7	7
SCHRPT1	39	42	7	20
SCHRPTA	42	45	7	20
SCHTOLD	31	34	7	3
SCHRPT	28	34	4	3
SETDELA	37	62	7	7
SETDELAY	37	63	7	20
SFCRANK	24	24	7	20
SFCRANK	24	24	7	46
SFCRANK	24	24	7	46
SFCSTAT	30	38	7	82

Name	LINES	LOC1	function points	cplx5
SILOGHD	71	72	7	82
SMFMLST	40	52	7	46
SROLLCT	28	38	5	7
SROLCT1	28	38	5	7
SSACMAY	32	77	5	46
SSACMAY	58	130	7	82
SSACTMAY	136	208	7	82
SSCHK1	50	54	7	82
SSCHK1	52	56	7	82
SSCHK2	37	46	7	20
SSCHK3	41	52	7	20
SSCHK3	41	52	7	20
SSCHKAJ	37	46	7	46
SSFMLST	40	55	7	20
SSFMLST	45	62	7	46
SSFTMAY	32	77	5	46
SSFTMAY	82	154	7	82
SSGFORM	22	29	5	3
SSGRPT1	42	63	5	82
SSGRPT2	48	95	5	82
SSLOGHD	69	69	7	82
SSLOGHD	60	60	7	82
SSLOGHD	60	61	7	82
SSLOGRP	59	60	7	82
SSMAYRO	98	170	7	82
TEST	20	20	4	3

A5.1 Non-Statistical Reports

Name	Data elements	Line types	Relations
ADLBLCR	3	2	6
ADLBLCR	3	2	6
ADFTSFC	28	2	8
ADFTSFC	13	11	7
ADFTSFC	30	5	9
ADFTSFC	13	6	7
CATREP	3	1	1
CHECKBT	15	1	3
CLASSCH	8	1	4
CLASSLS	7	2	3
COD	4	1	2
COD3	19	17	2
COMTRPT	7	7	2
COUNTGU	11	9	2
CRSHQWD	6	3	2
CRCHQWD	11	0	2
CRDWD	14	8	7
CRNOFMG	5	1	1
CRSCH	3	1	3
CRSLIST	12	8	6
CRSLIST	12	8	9
CRSLIST	12	8	6
CRSLTES	12	8	9
CRSMKFI	4	2	2
CRSNME	3	1	2
CRSUBJ	12	8	6
CRSWDLH	14	8	7
CRSWDLH	14	8	7
CRZIP	3	1	2
CS/SS/5	18	9	4
CSSS80F	15	13	5
CSSS80	15	10	5
DATACHK	24	4	3
DATACHK	24	4	3
DATACHK	14	4	6
DEANZ	4	1	2
DEFAULT	16	9	4
DEFAULT	16	9	4
DEFAULT	16	8	4
DJCHECK	14	10	5
DJCHECK	21	9	7
DJFMLST	17	8	5
DJINSLS	7	5	2
FORMLIST	9	3	2
FORMLIST	10	3	2
FPLOGCT	15	17	5
FTSSG	10	3	2
FTSSG	10	3	2
IXMRANK	16	10	5
IXMRANK	15	10	5
LBLIST	8	2	1
LBLLIST	8	2	1
LBLNZCH	11	2	1

Name	Data elements	Line types	Relations
LOGCT1	8	5	3
LOGCT1	12	5	4
LOGCT2	7	5	2
LOGCT2	7	5	2
LOGCT3	9	5	4
LOGCT4	20	8	4
LOGCTJH	14	17	5
LOGHDR1	34	10	6
LOGHDR5	27	0	8
MSCRSLS	12	10	6
MSFMLST	20	8	5
MWLOGHD	32	1	10
NMEDIST	7	4	1
PENDR3	17	15	1
PENDRPT	7	4	1
PENDRPT	7	4	1
PROLLCT	12	8	2
PROLLCT	11	8	2
PROLLCT	11	7	1
PROLLCT	11	8	1
PROLLCT	13	7	1
PROLLCT	13	7	1
PRFCMAY	15	15	2
PTFCMAY	40	28	2
PTFMLST	11	10	4
PRLOGHD	28	1	9
PTMAYRO	11	16	2
PTNO2DE	6	3	1
REPORT1	6	8	1
ROLLCT1	11	4	2
SCHLILST	4	3	1
SCHROL1	6	4	4
SCHROL2	19	2	5
SCHROL	6	2	3
SCHROL	14	3	5
SCHROLL	15	3	4
SCHRPT1	14	2	5
SCHRPTA	16	2	5
SCHTOLD	8	4	4
SCHRPT	12	10	1
SETDELA	19	9	5
SETDELAY	19	9	5
SFCRANK	3	2	6
SFCRANK	3	2	6
SFCRANK	3	2	6
SILOGHD	33	2	12
SMFMLST	13	10	5
SROLLCT	11	8	2
SROLCT1	11	8	2
SSACMAY	15	15	2
SSCHK1	18	4	10
SSCHK1	18	4	10
SSCHK2	12	7	4

Name	Data elements	Line types	Relations
SSCHK3	12	11	5
SSCHK3	13	11	5
SSCHKAJ	11	9	4
SSFMLST	16	8	5
SSFMLST	20	10	5
SSFTMAY	15	15	2
SSGFORM	8	5	2
SSGRPT1	24	26	1
SSGRPT2	39	28	1
SSLOGHD	33	2	11
SSLOGHD	29	2	10
SSLOGHD	30	2	10
SSLOGRP	30	2	9
TEST	6	1	1

Name	Logic	Logical Reports	Sort Fields
ADLBLCR	53	1	3
ADLBLCR	52	1	3
ADFTSFC	71	2	2
ADFTSFC	82	1	4
ADFTSFC	73	2	3
ADFTSFC	82	1	4
CATREP	1	1	1
CHECKBT	14	1	1
CLASSCH	3	2	3
CLASSLS	0	2	1
COD	3	1	1
COD3	37	1	2
COMTRPT	10	1	1
COUNTGU	12	1	2
CRSHQWD	4	1	0
CRCHQWD	4	1	0
CRDWD	19	1	5
CRNOFMG	3	1	1
CRSCH	9	1	2
CRSLIST	6	1	5
CRSLIST	65	1	5
CRSLIST	4	1	4
CRSLTES	61	1	5
CRSMKF	61	1	3
CRSNME	0	1	0
CRSUBJ	7	1	5
CRSWDLH	14	1	5
CRSWDLH	19	1	5
CRZIP	5	1	1
CS/SS/5	7	3	4
CSSS80F	58	1	5
CSSS80	63	1	5
DATACHK	16	1	1
DATACHK	15	1	1
DATACHK	65	4	1
DEANZ	7	1	2
DEFAULT	18	1	3
DEFAULT	16	1	3
DEFAULT	13	1	2
DJCHECK	20	1	4
DJCHECK	31	3	7
DJFMLST	43	2	3
DJINSLS	6	1	2
FORMLIST	3	1	2
FORMLIST	3	1	2
FPLOGCT	49	1	3
FTSSG	7	1	2
FTSSG	7	1	2
IXMRANK	17	1	2
IXMRANK	17	1	4
LBLIST	1	1	1
LBLLIST	1	1	5
LBLNZCH	8	1	2

Name	Logic	Logical Reports	Sort Fields
LOGCT1	5	1	1
LOGCT1	16	1	1
LOGCT2	5	1	1
LOGCT2	6	1	1
LOGCT3	5	1	1
LOGCT4	12	1	3
LOGCTJH	51	1	2
LOGHDR1	21	1	4
LOGHDR5	58	3	4
MSCRSLS	6	1	5
MSFMLST	43	1	3
MWLOGHD	63	3	5
NMEDIST	7	1	1
PENDR3	7	1	1
PENDRPT	2	1	1
PENDRPT	1	1	1
PROLLCT	7	1	1
PROLLCT	7	1	1
PROLLCT	16	1	1
PROLLCT	21	1	1
PROLLCT	24	1	1
PROLLCT	24	1	1
PRFCMAY	46	1	2
PTFCMAY	86	1	1
PTFMLST	22	2	3
PRLOGHD	62	3	5
PTMAYRO	33	1	2
PTNO2DE	7	1	0
REPORT1	18	1	2
ROLLCT1	7	1	1
SCHLILST	0	1	1
SCHROL1	20	2	2
SCHROL2	21	3	4
SCHROL	2	2	1
SCHROL	21	3	3
SCHROLL	15	3	3
SCHRPT1	21	3	2
SCHRPTA	21	3	2
SCHTOLD	5	2	4
SCHRPT	0	1	0
SETDELA	13	1	5
SETDELAY	16	1	5
SFCRANK	21	1	2
SFCRANK	28	1	2
SFCRANK	27	1	2
SILOGHD	82	4	6
SMFMLST	25	2	5
SROLLCT	10	1	1
SROLCT1	7	1	1
SSACMAY	46	1	2
SSCHK1	79	4	1
SSCHK1	79	4	1
SSCHK2	22	2	5

Name	Logic	Logical Reports	Sort Fields
SSCHK3	25	2	5
SSCHK3	25	2	5
SSCHKAJ	28	2	2
SSFMLST	20	2	3
SSFMLST	43	2	3
SSFTMAY	46	1	2
SSGFORM	3	1	2
SSGRPT1	74	1	0
SSGRPT2	111	1	2
SSLOGHD	70	4	6
SSLOGHD	68	3	5
SSLOGHD	68	3	5
SSLOGRP	67	3	5
TEST	0	1	5

Name	Control Breaks	cplx3	cplx5	LOC
ADLBLCR	2	60	68	78
ADLBLCR	2	60	28	77
ADFTSFC	0	60	68	125
ADFTSFC	2	60	68	127
ADFTSFC	1	60	68	134
ADFTSFC	2	60	68	125
CATREP	0	4	3	18
CHECKBT	0	17	17	46
CLASSCH	0	4	3	32
CLASSLS	0	4	3	26
COD	1	4	3	23
COD3	2	60	28	119
COMTRPT	2	17	7	50
COUNTGU	3	17	7	50
CRSHQWD	0	4	3	26
CRCHQWD	0	4	3	26
CRDWD	3	17	17	66
CRNOFMG	0	4	3	22
CRSCH	0	17	7	28
CRSLIST	2	4	7	49
CRSLIST	2	60	68	111
CRSLIST	2	4	3	47
CRSLTES	2	60	68	107
CRSMKFI	2	60	68	83
CRSNME	0	4	3	17
CRSUBJ	2	4	7	50
CRSWDLH	3	17	17	61
CRSWDLH	3	17	17	66
CRZIP	0	4	3	23
CS/SS/5	2	4	7	60
CSSS80F	2	60	68	110
CSSS80	2	60	68	112
DATACHK	0	17	17	59
DATACHK	0	17	17	58
DATACHK	0	60	68	114
DEANZ	0	4	7	26
DEFAULT	2	17	17	70
DEFAULT	2	17	17	68
DEFAULT	1	17	7	63
DJCHECK	2	17	17	70
DJCHECK	3	60	28	99
DJFMLST	2	60	28	97
DJINSLS	0	4	7	35
FORMLIST	1	4	3	31
FORMLIST	1	4	3	31
FPLOGCT	3	60	28	129
FTSSG	1	4	7	35
FTSSG	1	4	7	35
IXMRANK	1	17	17	69
IXMRANK	2	17	17	70
LBLIST	0	4	3	24
LBLLIST	0	4	3	24
LBLNZCH	0	17	7	34

Name	Control Breaks	cplx3	cplx5	LOC
LOGCT1	2	4	3	39
LOGCT1	2	17	17	56
LOGCT2	2	4	3	37
LOGCT2	2	4	7	38
LOGCT3	2	4	3	41
LOGCT4	4	17	7	62
LOGCTJH	3	60	28	130
LOGHDR1	1	17	17	82
LOGHDR5	0	60	68	112
MSCRSLS	2	4	7	51
MSFMLST	2	60	28	102
MWLOGHD	1	60	68	126
NMEDIST	1	4	7	33
PENDR3	0	4	7	54
PENDRPT	0	4	3	25
PENDRPT	0	4	3	24
PROLLCT	2	4	7	45
PROLLCT	2	4	7	45
PROLLCT	2	17	17	51
PROLLCT	2	17	17	58
PROLLCT	2	17	28	62
PROLLCT	2	17	28	63
PRFCMAY	1	60	28	123
PTFCMAY	2	60	68	156
PTFMLST	1	17	28	68
PRLOGHD	2	60	68	120
PTMAYRO	2	60	28	76
PTNO2DE	1	4	7	30
REPORT1	2	17	17	51
ROLLCT1	2	4	7	38
SCHLILST	0	4	3	18
SCHROL1	0	17	17	51
SCHROL2	0	17	17	72
SCHROL	0	4	3	30
SCHROL	0	17	17	70
SCHROLL	0	17	17	64
SCHRPT1	0	17	17	63
SCHRPTA	0	17	17	66
SCHTOLD	1	4	3	39
SCHRPT	0	4	3	34
SETDELA	3	17	7	75
SETDELAY	3	17	17	79
SFCRANK	1	17	17	45
SFCRANK	1	60	28	52
SFCRANK	1	60	28	51
SILOGHD	2	60	68	154
SMFMLST	2	17	28	77
SROLLCT	2	17	7	48
SROLCT1	2	4	7	45
SSACMAY	1	60	28	123
SSCHK1	0	60	68	133
SSCHK1	0	60	68	135
SSCHK2	2	17	28	68

Name	Control Breaks	cplx3	cplx5	LOC
SSCHK3	2	17	28	77
SSCHK3	2	17	28	77
SSCHKAJ	1	60	28	74
SSFMLST	1	17	17	75
SSFMLST	2	60	28	105
SSFTMAY	1	60	68	123
SSGFORM	1	4	3	32
SSGRPT1	1	60	68	137
SSGRPT2	3	60	68	205
SSLOGHD	1	60	68	139
SSLOGHD	1	60	68	128
SSLOGHD	1	60	68	129
SSLOGRP	1	60	68	127
TEST	0	4	3	20

A5.2 Statistical Reports

Name	Data elements	Line Types	Logical Reports
IXMST11	55	65	1
IXMST12	55	65	1
IXMSTAA	55	65	1
IXMSTAB	55	65	1
IXMSTAT	55	65	1
IXMSTAT	55	65	1
IXMSTA	55	65	1
SFCSTAT	9	9	1
SSACMAY	41	29	1
SSACTMAY	40	29	1
SSFTMAY	40	28	1
SSMAYRO	40	28	1

Name	Sort Fields	Control Breaks	cplx3
IXMST11	2	2	87
IXMST12	2	2	87
IXMSTAA	2	2	87
IXMSTAB	2	2	175
IXMSTAT	2	2	175
IXMSTAT	2	2	291
IXMSTA	2	2	291
SFCSTAT	1	1	291
SSACMAY	2	2	291
SSACTMAY	1	2	291
SSFTMAY	1	2	291
SSMAYRO	1	2	291

Name	logics	Relations	LOC
IXMST11	291	7	432
IXMST12	292	7	433
IXMSTAA	291	7	432
IXMSTAB	292	7	433
IXMSTAT	291	7	432
IXMSTAT	291	7	432
IXMSTA	290	7	431
SFCSTAT	201	7	239
SSACMAY	109	3	239
SSACTMAY	86	2	285
SSFTMAY	86	2	240
SSMAYRO	88	2	258

A6 ALL Reference Updates

Name	Control breaks	Data elements	Logical updates
Adlblcr	0	6	1
Agefix	0	2	1
Adfixrs	0	1	1
ajfixu	0	7	2
Bastid	0	2	1
Btchclr	0	1	1
Chngefm	0	3	1
Chngesf	0	1	1
Chngefn	0	3	1
Crsmk1	0	26	1
Crsmk2	0	26	1
Crsmk3	0	26	1
Crsmk4	0	26	1
Crssumu	0	13	2
Crssumu	1	13	1
Crsup1	0	2	1
Cslwk	0	5	2
Csstwd3	0	7	1
Cuscrse	0	0	2
Custrst	0	0	1
Custid	0	2	1
Lblcust	0	2	1
Logflag	0	2	1
Morelbl	0	2	1
Morelbl	0	2	1
Morelog	0	3	1
Ptpen1u	0	2	1
Sfcbset	0	3	1
SSduplu	0	8	1
SSfmfix	0	6	1
Stschnm	0	5	1
Xcrnum	1	2	1
Xfcform	1	4	1
Xfcform	1	4	1
Xrpstcr	0	2	1
Xrpstcr	0	0	2
Xrschst	0	0	2
Xrschst	0	3	1
Xrstcrs	0	2	1
Xrstcrs	0	0	2
Xrstln	0	0	1
Xrstln	0	0	1
Xrstsch	0	4	1
XRSTSCHU	0	4	1
Xrstsp	0	0	1
Xrstsp	0	0	2
Xrstsp	0	0	1

Name	Relations	Sort Fields	cplx3	cplx5
Adlblcr	3	0	10	10
Agefix	1	0	10	10
Adfixrs	1	0	3	3
ajfixu	3	0	10	10
Bastid	1	0	10	10
Btchclr	1	0	3	3
Chngefm	1	0	3	3
Chngesf	1	0	3	3
Chngefn	1	0	3	3
Crsmk1	4	0	88	88
Crsmk2	3	0	88	88
Crsmk3	4	0	88	88
Crsmk4	3	0	88	88
Crssumu	3	2	10	34
Crssumu	2	4	10	34
Crsup1	1	0	10	34
Cslwk	4	1	3	3
Csstwd3	1	1	10	10
Cuscrse	2	0	3	3
Custrst	3	0	3	3
Custid	1	0	10	34
Lblcust	1	0	10	34
Logflag	1	1	3	3
Morelbl	1	1	3	3
Morelbl	1	1	3	3
Morelog	1	1	3	3
Ptpen1u	1	0	10	38
Sfcbset	2	0	3	3
SSduplu	2	0	10	10
SSfmfix	2	0	3	3
Stschnm	2	0	3	3
Xcrnum	2	1	10	10
Xfcform	3	1	10	10
Xfcform	3	1	10	10
Xrpstcr	3	0	3	3
Xrpstcr	2	0	3	3
Xrschst	2	2	3	3
Xrschst	3	3	3	3
Xrstcrs	3	0	3	3
Xrstcrs	2	0	3	3
Xrstln	3	2	3	3
Xrstln	2	2	3	3
Xrstsch	3	0	3	3
XRSTSCHU	3	0	3	3
Xrstsp	3	0	3	3
Xrstsp	2	0	3	3
Xrstsp	3	0	3	3

Name	logics	LOC
Adlblcr	6	21
Agefix	6	19
Adfixrs	5	18
ajfixu	10	32
Bastid	10	23
Btchclr	3	18
Chngefm	4	16
Chngesf	3	16
Chngefn	3	15
Crsmk1	80	97
Crsmk2	96	112
Crsmk3	80	97
Crsmk4	96	112
Crssumu	29	52
Crssumu	25	42
Crsup1	28	39
Cslwk	4	26
Csstwd3	10	26
Cuscrse	0	18
Custrst	0	14
Custid	43	56
Lblcust	46	59
Logflag	5	20
Morelbl	3	19
Morelbl	3	18
Morelog	4	20
Ptpen1u	38	51
Sfcbset	3	14
SSduplu	6	22
SSfmfix	5	20
Stschnm	3	18
Xcrnum	6	24
Xfcform	18	37
Xfcform	19	38
Xrpstcr	4	20
Xrpstcr	0	18
Xrschst	0	18
Xrschst	3	20
Xrstcrs	5	20
Xrstcrs	0	18
Xrstln	0	16
Xrstln	0	16
Xrstsch	3	18
XRSTSCHU	3	18
Xrstsp	0	13
Xrstsp	0	18
Xrstsp	0	13

APPENDIX B

ALL TEST DATA

B1 ALL Test SCRMENUS

Appendix B Test Menus

Menu Number	choice	LOC	Pred	RE	in	abs	e^2
1	3	34	35.77	-0.05	1	0.0521	3.1329
2	3	44	35.77	0.19	2	0.187	67.7329
3	14	78	79.66	-0.02	3	0.0213	2.7556
4	6	45	47.74	-0.06	4	0.0609	7.5076
5	6	46	47.74	-0.04	5	0.0378	3.0276
		247	246.68	0.01	100%	0.3591	84.1566
				RE*		MRE*	mean
		mean		0.003		0.0718	16.83132
		49.4					RMS
							4.102599
							RMS*
							0.083049

B2 ALL Test Relations

Appendix B Test Relations

RELATION NAME	Data elements	Actual LOC	Predicted LOC
ANOMOLIES	5	6	10.77
BATCHMARK	6	7	12.65
BKUPSTUD	100	101	189.37
BatchSets	4	5	8.89
BtCntrol	2	3	5.13
CALENDAR	7	8	14.53
CNTTEST	3	4	7.01
CRSROLL	3	5	7.01
CRSWDLS	2	3	5.13
CSFOFORM	2	3	5.13
CLASSES	2	3	5.13
CRSROLL	2	4	5.13
DECHECK	1	2	3.25
DEPTMSG	8	9	16.41
DEPTBATCH	2	4	5.13
EPA	8	9	16.41
EICHRN	2	3	5.13
EXMCHRSN	2	4	5.13
FTJLUIDISCAT	2	3	5.13
GDACOUNT	14	15	27.69
GKEYS	2	4	5.13
GDSTUD	2	4	5.13
INACTSTUD	6	7	12.65
INSTPREXM	3	4	7.01
INACTSTQUAL	2	3	5.13
LOGOF	7	8	14.53
OMNZCS	6	7	12.65
PRCLASS	3	4	7.01
PTCOUNT	2	3	5.13
PTPRLBL	6	7	12.65
PWDTCH	2	3	5.13
QSSTUDENT	76	77	144.25
QSTQUAL	2	4	5.13
ROLLSTATS	178	179	336.01
SCHSTUDCRS	3	4	7.01
SFCDATA	8	9	16.41
SSDUPLICAT	4	5	8.89
STDCRSSP	15	16	29.57
STEXMSUB	16	17	31.45
SECSTUDCRS	4	5	8.89
STPSTSET	2	3	5.13
TCHLEAVE	10	12	20.17
TCHSBJ	3	5	7.01
WORKGD	2	3	5.13
WSRPREPROG	2	3	5.13
XTEACHER	45	46	85.97
		643	1168.46
		mean	pred mean
		13.97826	25.4013

B3 ALL Test Screens

Function number	FPs	Input data elements	Output data elements	menu FPs	total Fps	cplx3
1	7	10	12	3	10	38
2	7	5	13	3	10	38
3	3	12	1		3	0
4	6	13	1		6	0
5	3	13	1		3	0
6	3	3	0		3	0
7	5	4	4	3	8	0
8	3	3	2	3	6	17
9	3	7			3	0
10	4	3	3	3	7	17
11	4	3	3	3	7	17
12	5	2	2		5	0
13	3	2			3	0
14	4	3	3	3	7	0
15	3	5			3	0
16	3	7	0		3	0
17	3	12			3	0
18	5	4	5		5	17
19	4	4	3		4	38
20	4	1	3		4	38
21	4	4	4	3	7	38
22	7	1	33	3	10	38
23	7	12	8	3	10	38
24	3	6	4	3	6	38
25	7	13	7	3	10	38
26	4	4	4		4	38
27	4	4	4	3	7	38
28	7	9	8	3	10	17
29	7	7	10	3	10	38
30	7	1	12	3	10	17
31	7	2	20	3	10	38
32	6	7	4	3	9	38
33	4	2	5		4	17
34	5	5	5	3	8	17
35	5	2	10	3	8	17
36	4	1	2	3	7	17
37	4	1	5		4	0
38	4	1	2	3	7	0
39	4	2	1		4	0
40	3	7	0		3	0
41	3	5	0		3	0
	sum			sum		
	188			66		

Function number	LOC	logic	cplx5	cplx3	Data elements	relations	choice
1	251	179	95	38	22	10	3
2	174	113	95	38	18	9	3
3	36	0	0	0	12	1	0
4	45	0	0	0	14	3	0
5	43	0	0	0	14	1	0
6	22	0	0	0	3	1	0
7	37	0	0	0	8	5	2
8	44	15	11	17	5	1	3
9	30	0	0	0	7	1	0
10	49	14	11	17	6	2	2
11	47	13	11	17	6	2	2
12	35	7	11	0	4	4	0
13	19	0	0	0	2	1	0
14	36	8	11	0	6	2	2
15	26	0	0	0	5	1	0
16	30	0	0	0	7	1	0
17	38	0	0	0	12	1	0
18	63	24	23	17	9	6	0
19	31	3	0	38	7	3	0
20	33	6	11	38	4	0	0
21	51	18	23	38	8	3	2
22	116	45	41	38	34	6	3
23	126	64	95	38	20	4	2
24	97	61	41	38	10	1	2
25	107	42	41	38	20	8	2
26	62	28	23	38	8	1	0
27	70	33	41	38	8	2	2
28	98	28	23	17	17	7	2
29	138	35	41	38	17	7	3
30	84	27	23	17	13	5	2
31	121	37	41	38	22	7	2
32	72	31	41	38	11	4	2
33	66	25	23	17	7	2	0
34	80	23	23	17	10	8	2
35	66	15	11	17	12	3	2
36	26	1	0	17	3	1	2
37	36	7	11	0	6	3	0
38	26	1	0	0	3	1	2
39	28	7	11	0	3	1	0
40	30	0	0	0	7	1	0
41	26	0	0	0	5	1	0
	2615						
	63.78						

B4 ALL Test Reports

FUNCTION Number	Data elements	Relations	Sort Fields	Line Types	LOGICS	cplx3	cplx5	LOC
1	59	5	3	24	121	70	82	236
2	42	1	2	29	113	70	82	242
3	55	7	2	68	295	70	82	447
4	62	1	0	22	234	70	82	327
5	62	4	3	26	106	70	82	222
6	16	9	5	10	38	20	46	101
7	12	3	3	12	29	20	46	78
8	30	4	2	16	87	70	82	169
9	23	3	2	6	83	70	82	135
10	20	5	3	8	16	20	20	82
11	12	4	5	10	9	5	7	56
12	15	5	2	14	17	20	20	43
13	24	6	2	14	21	20	20	89
14	23	5	3	15	23	20	20	97
15	5	2	0	1	7	5	7	31
16	5	2	0	1	3	5	3	24
17	11	4	4	7	3	5	3	45
18	6	1	0	5	1	5	3	35
19	21	5	2	14	15	20	7	79
20	10	3	1	14	14	20	7	64
21	15	5	5	10	11	5	7	69
22	11	5	4	7	11	5	7	54
23	6	4	1	6	18	20	20	55
24	11	4	4	11	6	5	7	55
25	11	5	6	7	27	20	46	82
26	8	3	4	7	20	20	20	59
27	18	5	3	17	62	70	46	131
28	17	6	5	7	50	70	82	116
29	16	5	5	13	37	70	46	104
30	42	1	2	29	113	70	82	279
31	42	2	2	31	116	70	82	280
32	51	1	0	27	99	70	82	199
33	11	5	12	5	20	20	20	72
34	3	6	2	2	17	20	20	48
35	22	2	2	13	22	20	20	87
36	37	3	3	5	167	70	82	234
37	13	6	6	3	28	20	46	78
38	20	6	3	10	37	20	46	106
39	18	5	4	10	45	70	46	104
40	21	7	5	14	299	70	82	363
41	32	6	2	6	165	70	82	230
								Total 5407
								Mean 132

FUNCTION Number	Function Points
1	7
2	5
3	7
4	5
5	7
6	7
7	5
8	7
9	7
10	7
11	7
12	7
13	7
14	7
15	4
16	4
17	7
18	4
19	7
20	5
21	7
22	7
23	5
24	7
25	7
26	5
27	7
28	7
29	7
30	5
31	7
32	5
33	7
34	5
35	7
36	7
37	7
38	7
39	7
40	7
41	7
	Total
	260

B4.1 Non-Statistical Reports

Appendix B Non-stats Reports

Report No	Data elements	Relations REL	Sort Fields	Line Types	LOGICS	cplx3	CPLX5	LOC
5	62	4	3	26	106	61	68	222.00
6	16	9	5	10	38	61	28	101.00
7	12	3	3	12	29	61	68	78.00
8	30	4	2	16	87	61	28	169.00
9	23	3	2	6	83	61	68	135.00
10	20	5	3	8	16	17	17	82.00
11	12	4	5	10	9	17	7	56.00
12	15	5	2	14	17	17	17	43.00
13	24	6	2	14	21	17	17	89.00
14	23	5	3	15	23	17	28	97.00
15	5	2	0	1	7	4	7	31.00
16	5	2	0	1	3	4	3	24.00
17	11	4	4	7	3	4	3	45.00
18	6	1	0	5	1	4	3	35.00
19	21	5	2	14	15	17	17	79.00
20	10	3	1	14	14	17	17	64.00
21	15	5	5	10	11	17	7	69.00
22	11	5	4	7	11	17	7	54.00
23	6	4	1	6	18	17	17	55.00
24	11	4	4	11	6	4	7	55.00
25	11	5	6	7	27	61	28	82.00
26	8	3	4	7	20	17	17	59.00
27	18	5	3	17	62	61	68	131.00
28	17	6	5	7	50	61	28	116.00
29	16	5	5	13	37	61	28	104.00
33	11	5	12	5	20	17	17	72.00
34	3	6	2	2	17	17	17	48.00
35	22	2	2	13	22	17	28	87.00
36	37	3	3	5	167	61	68	234.00
37	13	6	6	3	28	61	28	78.00
38	20	6	3	10	37	61	28	106.00
39	18	5	4	10	45	61	28	104.00
40	21	7	5	14	299	61	68	363.00
41	32	6	2	6	165	61	68	230.00
								3397
								mean
								99.91

B4.2 Statistical Reports

Appendix B Statistical Reports

Function Number	Data elements	Relations	Sort Fields	Line Types	LOGICS	cplx3	TOTLOC
1	59	5	3	24	121	200	236
2	42	1	2	29	113	200	242.00
3	55	7	2	68	295	291	447.00
4	62	1	0	22	234	291	327.00
30	42	1	2	29	113	200	279.00
31	42	2	2	31	116	200	280.00
32	51	1	0	27	99	86	199.00
							2010
							mean
							287.14

B5 ALL Test Updates

Appendix B Test Updates

Function Number	LOC	Data elements	Logical Updates	Control Breaks	Logic	cplx3	cplx5
1	24	8	1	0	8	29	10
2	17	2	1	0	2	0	3
3	12	1	1	0	1	0	3
4	46	16	1	0	28	29	10
5	34	13	1	0	21	29	10
6	39	13	2	0	18	29	10
8	41	11	1	0	24	29	10
9	22	4	1	0	7	29	10
10	79	12	1	0	63	29	80
11	43	10	1	0	26	29	10
12	21	6	1	0	6	5	5
13	12	0	1	0	0	0	0
14	22	3	2	0	3	0	3
15	31	2	1	0	18	29	10
16	27	4	2	1	3	0	0
17	29	7	1	0	13	29	10
18	26	3	2	1	3	0	3
19	26	5	1	0	12	29	10
20	19	4	1	0	4	5	3
21	13	1	1	0	1	0	3
22	25	2	1	0	11	29	10
23	22	8	1	0	8	29	10
24	18	2	1	0	5	5	5
25	35	10	1	0	21	29	10
26	17	2	1	0	2	0	3
27	16	2	1	0	2	0	3
28	15	2	1	0	3	0	3
29	66	8	6	0	11	29	10
	Total 797 Mean 27.48						

APPENDIX C

OTHER SOFTWARE DEVELOPMENT TECHNOLOGIES

C1 Menus for Three Technologies

Name	Choices	AREV	Lines of code Informix	COBOL
ML1	5	7	23	86
ML2	8	10	31	93
ML3	5	7	20	84
ML4	8	10	33	96
ML5	6	8	27	86

C2 COBOL and INFORMIX Input/Updates

NAME	Relations/Files	Data Elements	Informix LOC	COBOL LOC
SSU1	1	2	69	126
SSU2	1	2	101	208
SSU4	2	4	84	184
SSU5	1	4	85	163
SSU6	4	3	100	202
SSU7	1	2	66	141
NTS1	2	4	102	265
NTS3	1	4	108	250
MSL1	3	4	115	219
MSL4	3	4	116	263
MSL6	3	6	124	259
			total	total
			1070	2280
			mean	mean
			97.273	207.273

C3 COBOL and INFORMIX Reports/Inquiries

	Relations/Files	Data elements	Informix LOC	COBOL LOC
NTS2	1	4	68	172
SSU3	1	2	60	148
MSL7	3	5	110	308
MSL5	3	5	105	307
MSL2	3	3	104	307
MSL3	4	5	104	303
PO1	3	5	71	264
NTS4	1	4	74	174
INQ2	1	4	72	147
INQ1	1	2	70	126
INQ3	1	3	74	146
			total	total
			912	2402
			mean	mean
			82.91	218.36

C3.ADVANCED REVELATION Forms

	Relations	Data elements	Logics	AREV LOC
SSU1	1	2	0	48
SSU2	1	2	0	59
SSU4	2	4	0	82
SSU5	1	4	0	81
SSU6	4	3	0	68
SSU7	1	2	0	55
NTS1	2	4	0	92
NTS3	1	4	0	92
MSL1	3	4	0	75
MSL4	3	4	2	116
MSL6	3	6	2	111
MSL7	3	5	1	81
PO1	3	5	1	65
MSL2	3	3	2	92
MSL3	4	5	2	93
MSL5	3	5	1	81
				total
				1291
				mean
				80.6875
				0.8125

C3.ADVANCED REVELATION Pop-Ups

Name	Data elements	AREV LOC
INQ1	2	8
DAT	1	7
INQ2	4	10
ITS	2	8
WDL	2	8
INQ3	3	9
SOH	2	8
RPL	4	10
SLD	1	7
		total
		75
		mean
		8.33

APPENDIX D

Relationship of the Generic Model to the Bailey and Basili Meta-Model

In 1981 Bailey and Basili [BAIL81] suggested a meta-model for software development resource expenditures. The generic model described in this thesis shows similarities to this model in that both are meta-models requiring tailored instantiation in order to be applied by users in their own organizations. Different users of these models will develop different prediction equations tailored to their own development environment based on a few important size (or cost) drivers. However on another level the models are quite different.

The Bailey and Basili model computes a standard effort which is then adjusted by factors from the environment which account for variations from this standard effort. The generic model on the other hand sizes each system component individually within its appropriate component type and then sums the individual contributions. There is some similarity in the treatment of adjustment factors, which are critically surveyed in section 4.5.2, but are not used in the instantiations in the thesis. The resemblance in overall approach between the Bailey and Basili meta-model and the generic model is much the same as that between the Bailey and Basili model and any generalized FPA-like method which needs to be tailored to each particular environment.

To instantiate the generic model within a particular environment requires the user, briefly, to do the following:

choose a size measure
form component partitions based on software technology
set up a vector of size driver variables for each component type
derive size estimation equations for each component type by multiple regression or some other technique; individual component estimates will be summed during actual use to get system estimates
assess the need for overall adjustment factors; if needed, choose an appropriate set from a number of possible adjustment approaches, some multiplicative, some additive.

To instantiate the Bailey and Basili model within a particular environment requires the user, briefly, to do the following:

choose and define measures of size and effort
select the form of the baseline equation to get a measure of standard effort; this is found by fitting a curve through a scatterplot of effort versus size data
calculate an initial base-line for use in the model using multiple regression or some other method
collect factors which account for projects that are abnormal
choose a set of factors to explain these variations
estimate the necessary factor values by multiple regression
predict the deviations of the points from the computed base-line
convert the error ratio back to multiplicative factors and incorporate them into the standard effort equation to give a revised effort equation.

Form of Regression Equations

Some comment should be made regarding both the form of the component size estimation models and the occurrence of squared terms within the resulting regression equations.

The additive form of component size estimation models can be supported for two reasons.

a) The definition (programming) of a component in the 4GL used is done in two parts, a non-procedural part and a procedural logic part. The non-procedural part for a component definition is made up of a series of forms (Table 5.2 lists the forms required for the various component types). The non-procedural forms comprise two parts, a fixed part with some required and some optional inputs, and a variable part. The number of lines entered in the variable part depends on the complexity of the components being defined including the number of logical components contained within a single physical component (essentially a nesting level in the range 0-4), the number of data elements used, the number of relations accessed, etc. Typically a form is concerned with one aspect of a component, e.g. relations, data elements or line types, not with several, except perhaps implicitly. The logic part of a component is programmed in a procedural language and the number of LOC in this part varied from 0-300 LOC with a median of 4 LOC. The total of all

logic lines made up a little under 40% of the system on which the development of the model is based. The total lines of code used is the sum of the lines in the non-procedural forms and the logic. The additive nature of the regression model for each component type bears a strong correspondence to the way in which a set of different forms, whose separate sizes are affected largely by different single size drivers, are combined in an essentially additive way to form a complete component.

b) Median polish [HOAG83, HOAG85], a technique for investigating an appropriate form of a model to use with a particular set of data, was used with both screen and report data. This investigation showed no pattern in a diagnostic plot of residuals to suggest that an additive fit was inappropriate or that either a multiplicative or other form of non-additive fit would be more appropriate.

Squared data elements terms are used in Phase1 and 2 equations for both screens and reports while a squared relations term occurs in the phase 1 equations for screens. It should be noted that the use of these squared terms did not improve the model's R^2 greatly but did give a marginally better fit. Where these squared terms are included in the equations it is for the following reasons:

Without them the residuals from the regression equations tended to form a curved band suggesting that an improved fit would result from the inclusion of a quadratic expression of one or more of the terms [HOAG83]. In all cases where a squared term was significant in a regression, the corresponding linear term was not significant and was therefore removed.

a) Screens

Phase1 screen equations contain both (data elements)^2 and relations^2, while phase2 equations (including a complexity term), drop out relations^2 and use relations instead. This suggests that relations^2 in the phase 1 screen equations is accounting for some of the procedural logic which is explained in phase2 by cplx5. This would not be surprising as much of the logic is concerned with manipulating data which has been read from or is to be written to relations. The amount of logic concerned directly or indirectly with file (relation) processing tends to grow quadratically with the number of relations involved.

The squared data elements term occurs in both phase 1 and phase2 equations for possibly more complex reasons. Introduction of a cplx5 term in the phase2 equations decreases the (data elements)^2 coefficient to nearly half its original value but it still remains a statistically

significant factor though it is actually accounting for only very few lines of code except at quite high data element levels. The change in the (data elements)^2 coefficient between phase 1 and phase2 suggests that for the phase 1 equation, (data elements)^2 is accounting for some of the procedural logic. However data elements contribute to the size of a screen in several ways:

in their definition which occurs in the fields form;

in their prompts and labels (which will actually appear on the screen), in the screen definition form;

in the size of the screen characteristics form which is likely to increase considerably with increasing numbers of data elements, the increase often resulting from having several logical screens within the one physical screens;

in the number of procedural logic items required which increases with the number of data elements processed.

It is this multiplicity of effect that is likely to be responsible for data elements appearing as a squared term in the screen equations.

b) Reports

The (data elements)^2 term is present in report equations at both phase 1 and 2. The introduction of a cplx3 term reduces the coefficient in the phase2 equation. The reasons for inclusion of a (data elements)^2 term in the report equations are similar to those described above for screens.

During data exploration, a number of different models were tried, including other powers of additive terms, first order interactions, and some multiplicative models. None of these gave fits with R^2 values as good as the models finally adopted.

Both the statistical analysis and an intuitive understanding of the data in relation to the model support the validity of the model as does the fact that it is a very good fit to the data, predicting the size of the test system within 97% of actual LOC at phase 1 and 96% of actual at phase2.

Additional References

HOAG83 David Hoaglin, Frederick Mosteller and John Tukey, Editors, Understanding Robust and Exploratory Data Analysis, John Wiley and Sons, New York, 1983,

HOAG85 David Hoaglin, Frederick Mosteller and John Tukey, Editors, Exploring Data Tables, Trends, and Shapes, John Wiley and Sons, New York, 1985.