# 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:

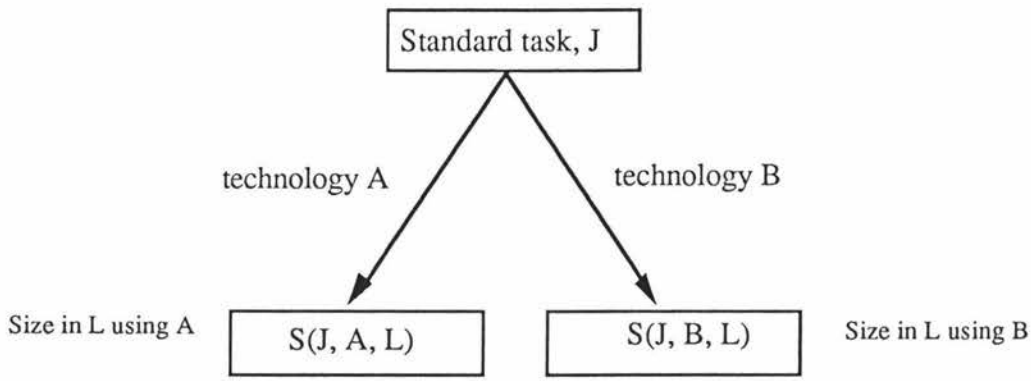
(1) to measure developer and/or managerial effectiveness within a defined software development environment (developer productivity)

(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.

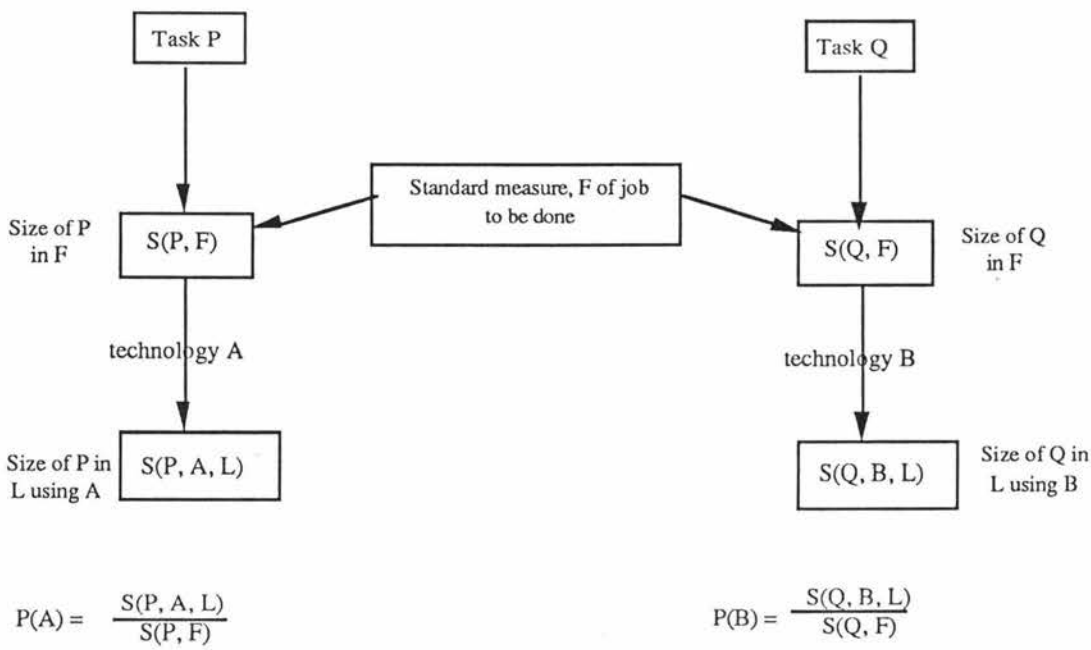



**Figure 1.2 The Standard Measure Approach to Software Productivity Measurement**

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.