ESTIMATING THE EFFORT IN THE EARLY STAGES OF SOFTWARE DEVELOPMENT

Zeeva Levy

London School of Economics and Political Science

Submitted in fulfilment of the requirements for the award of the degree of Doctor of Philosophy of the University of London.

July, 1990

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Estimates of the costs involved in the development of a software product and the likely risk are two of the main components associated with the evaluation of software projects and their approval for development. They are essential before the development starts, since the investment early in software development determines the overall cost of the system. When making these estimates, however, the unknown obscures the known and high uncertainty is embedded in the process. This is the essence of the estimator's dilemma and the concerns of this thesis.

This thesis offers an Effort Estimation Model (EEM), a support system to assist the process of project evaluation early in the development, when the project is about to start. The estimates are based on preliminary data and on the judgement of the estimators. They are developed for the early stages of software building in which the requirements are defined and the gross design of the software product is specified. From these estimates only coarse estimates of the total development effort are feasible. These coarse estimates are updated when uncertainty is reduced.

The basic element common to all frameworks for software building is the activity. Thus the EEM uses a knowledge-base which includes decomposition of the software development process into the activity level. Components which contribute to the effort associated with the activities implemented early in the development process are identified. They are the size metrics used by the EEM. The data incorporated in the knowledge-base for each activity, and the rules for the assessment of the complexity and risk perceived in the development, allow the estimation process to take place. They form the infrastructure for a 'process model' for effort estimating.

The process of estimating the effort and of developing the software are linked. Assumptions taken throughout the process are recorded and assist in understanding deviations between estimates and actual effort and enable the incorporation of a feedback mechanism into the process of software development.

These estimates support the decision process associated with the overall management of software development, they facilitate management involvement and are thus considered as critical success factors for the management of software projects.

To my family, who went with me all the long way, with love.

ACKNOWLEDGEMENTS

On my completion of this research, I would like to thank the staff and my colleagues in the Department of Information Systems at the London School of Economics. Particular thanks are extended to my supervisor, Professor Ian Angell, who was a constant source of advice and encouragement. His support helped me shape many of the ideas presented in this thesis, to conduct my research successfully and to bring it to completion.

My sincere thanks to the many individuals in Israel, the UK and the Netherlands for their time, co-operation and contribution to this research by acting as facilitators, completing the questionnaires, participating in the walkthrough sessions, commenting and advising.

I extend special thanks to my friend and colleague Dr. Edgar Whitley, who worked with me to build the prototype. Edgar made every effort to transform my ideas into a working system and as a result could give help and advice in many other aspects of the research, thank you Edgar.

Last, but by no means least, I would like to thank my family for all the support, encouragement and love. This thesis is dedicated to them.

TABLE OF CONTENTS

PART I

LIST OF APPENDICES 305

LIST OF FIGURES

LIST OF TABLES

In **Appendix 7C** :

Chapter 1

THE PROBLEM DOMAIN

1.1 INTRODUCTION

This chapter discusses the motivation for estimating the effort needed for software development and the problems associated with this process. The various approaches, models and tools currently available are analysed. This establishes the foundation for further research by proposing the infrastructure for a process model for the task of forecasting the development effort. The results will assist development project team members, managers and users of software, by linking between the process of software development with that of estimating the effort needed for software building.

How does this research fit into the broader area of software engineering and management science research? The processes of software development and effort estimation are strongly interconnected. Software strategy decisions affect the effort estimation process, while the estimates resulting from the effort estimation process affect decisions related to the development process. Cost estimation is thus a major layer in the economic evaluation of a new or an upgraded software product. The ability to evaluate software cost is dependent on fundamental knowledge stemming from computer science and management science, incorporating various disciplines such as information systems, organisational behaviour and psychology.

1.2 DEFINITION OF SOFTWARE AND SOFTWARE ENGINEERING

Let us begin by defining software, the product and the discipline used for the process of developing the software product, for which estimates are needed. This thesis considers software in a broad sense. Webster's New Collegiate Dictionary's definition of software is a good starting point:

> *Software is the entire set of programs, procedures and related documentation associated with a system, especially, computer system* [Web79].

The emphasis here is on the words **system** and **computer system.** Webster defines a system as:

> *A group of interacting bodies under the influence of related forces,* or *the body considered as a functional unit* [Web79].

Within this thesis, the key terms are **interacting bodies** and **functional units.** The interacting bodies are processed for attaining an end, a functional unit. In emphasising the functional unit, it is suggested that software includes, both the application software itself and the operational configuration on which the application software is built. A computer based system is defined as:

> *A set or arrangement of elements organised to accomplish some method, procedure or control by processing information* [Web79].

The elements involved in this arrangement are, procedures, documentation, hardware, systems, data-bases, software and people. This definition is quite broad and includes more than just computer programs. However, even this broad definition may not suffice as software itself is a very general term. For instance, considering software projects and systems, software products and software support, do we really mean the same when using each of these terms? The answer is, usually not. Fox [Fox82] states:

> *Software is too broad a word. It is generic, like the word 'animal' which can be a pet or a cat or an 800 pound Polar bear. Yet, people talk about software as*

though it were a thing, or a uniform body of things. It is everything but.

The term software engineering was introduced at the 1968 NATO Workshop devoted to the issue, at Garmisch, West Germany [Nau69]. The software engineering concept evolved further in the 1970's with Fritz Bauer, applying the stronger disciplines of engineering (in contrast to 'art') to the software development process. Bauer defined software engineering as:

> *The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works efficiently on real machines* [Bau72].

This definition encompasses the key issues at the heart of all definitions of the engineering discipline. 'Sound engineering principles' for the development process should result in an economical, reliable and functional product. The term 'sound engineering principles' also includes managerial considerations.

The fundamental objective of software engineering is to produce a quality product and to reduce the severity of possible failures in software development. The software engineering concept encompasses the key factors of methods, tools and procedures in support of all principle stages of software development. This enables the manager of software development to gain control over the process and provides the system developer with a foundation for building software. Based on these concepts, more comprehensive definitions have been proposed, all reinforcing the importance of engineering discipline in software development. Boehm's [Boe81] definition emphasises the management of expectation and the necessity of satisfying human need, which means that software is a working product only when it satisfies a set of requirements:

> *Software Engineering is a application of science and mathematics, by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation* [Boe81].

1.3 SOFTWARE ENGINEERING AS AN EDUCATIONAL SUBJECT

Researchers and practitioners, including Freeman [Fre76], Wegner [Weg80], Jensen and Tonic [Jen79], Boehm [Boe81], Sommerville [Som85] and Macro and Buxton [Mac87], view software engineering as a branch of engineering. Macro and Buxton [Mac87] question whether or not software engineering should be taught in educational establishments, and if so how it should be discriminated from computer science. Jensen and Tonic suggest that a software engineering curriculum, as a professional and as an academic subject, should be composed of the following primary areas: management science, engineering fundamentals, computer science, physical science, communication skills and project laboratory work.[1] They propose to place cost estimation under the umbrella of management science and in particular under the topic of cost analysis.

Yet, it is controversial whether estimating software cost should be taught only as a part of cost analysis. Cost estimates support the decision process associated with the project definition and the management of software development. Furthermore, the parties involved in this decision process come from various parts of the organisation and may have conflicting objectives and views of the solutions offered for a given set of requirements. Complexity and uncertainty, caused by a wide range of problems, are also associated with the development of software and therefore with the estimation process. The reasons for complexity in software development might be logical and/or technical. Problems stem from the nature of the application being developed, or external factors such as the organisational and the technical environments. Modelling software costs is dependent on understanding the broad areas of management science and computer science. However, modelling of software costs interacts with other disciplines in management science, particularly organisational, cognitive psychology and decision theories. The subject of estimating the effort required for software development would undoubtedly benefit from interdisciplinary treatment.

---

1. This work was supported by the Institute of Electrical and Electronic Engineers, Inc. (IEEE).

1.4 THE NEED FOR EFFORT ESTIMATION

> *Planning and forecasting is imperfect work beset with uncertainties of various kinds, yet it is essential to keep the gap between disaster and simple variance wide enough to enable progress to be made in some orderly way* [Blu69].

The incentive for estimating the effort needed in software development arises from the need for planning. Forecasting is a difficult management task, since uncertainties of all kinds are involved. Planning is the process of setting formal guidelines and constraints for managerial action. Its purpose is to show how to act instead of react, to provide adaptable methods and to overcome the alarm syndrome that prevails today. Management of software is still a matter of personal style and individual experience. Management, from a software engineering viewpoint, is primarily the management of the design process. The process is highly creative knowledge work, yet it must still be estimated and scheduled so that the various life cycle activities can be co-ordinated and integrated into an harmonious result.

Knowing the estimated cost of software development assists us in the processes of justifying the cost for a software project, analysing realistic tradeoffs, planning resource requirements, designing-to-cost and/or designing-to-schedule, and controlling the development process.[2]

The need to estimate the effort required for software development exists before and during the development process, therefor, the cost analysis of a project should be a continuing activity throughout the project life cycle. Yet, as decisions made early in the software development determine the cost of the software life cycle, an estimate of the total effort required to build a software product, its schedule and cost, is essential before any investment decision is made. Estimating the effort required to develop a software product is a major factor in evaluating a potential project. Labour is not only the most expensive resource, it is a scarce resource: there is a shortage of high quality, experienced people. We can't

---

2. This is an iterative procedure. It consists of design and estimation activities and aims to ensure design-to-cost and design-to-schedule. This process also enables us, if it is at all necessary, to design only the minimum to satisfy a set of requirements stated by the customer.

"...indefinitely, add people and get the job done faster" [Nor63], nor are *"Manpower and time interchangeable"* [Bro75]. Knowledge about the behaviour of this valuable resource assists management in prioritising both alternative solutions and development strategies, in which the estimated effort differs meaningfully.

The timing of the installation of a new project might be of critical importance to the user organisation as far as changes in the economic feasibility of the project are concerned. Software users are often more concerned about predictability and control over software costs and schedule than they are about the absolute values of the costs and schedule. Good estimates enable management to synchronise their software development with other critical development, such as major changes in their services.

Knowledge about the estimated effort and the development schedule, for each development phase, is essential through the project life cycle. This knowledge enables managers to control the development process by tracing the project status with regard to two important factors: the effort and schedule associated with the process.

However, the degree of understanding of the project under development varies throughout the life cycle. The uncertainties in factors influencing the estimating process are also reduced as the development process continues. The nature of the estimates differ for each type of system, as well as in each phase of the software development life cycle, according to the estimation objectives and the maturity level of the project. Therefore, although an interdependent estimation model for the entire life cycle is claimed to be of benefit, this research will question whether it is reasonable (or worthwhile) to explore such a model. This research takes the attitude that each software development phase should employ different estimating procedures.

1.5 THE PROBLEM DOMAIN

The 1980's have often been called the information decade. The trends established throughout this decade are certain to continue into the 1990's and probably well

into the next century. The implications of these trends on management of software development should be understood. The convergence in the late 1970's of computer technology (hardware & software) with communications has resulted in the software development environment becoming far more complex. Man-power costs have increased continually while the management skills needed to control this complex process, which is technically oriented and heavily affected by human and organisation behaviour factors, have remained scarce.

The computer industry has seen a dramatic rise in the cost of software relative to hardware. Hardware costs have been declining significantly while software costs have been increasing. Typically, at the time when only 2% of the project costs have been spent, a commitment already exists for 70% of the software and hardware costs, as shown in Figure 1.1. Whilst this percentage varies slightly, the conclusion is well supported. Therefore, decisions made at the outset of a project will significantly and unalterably shape the system cost.

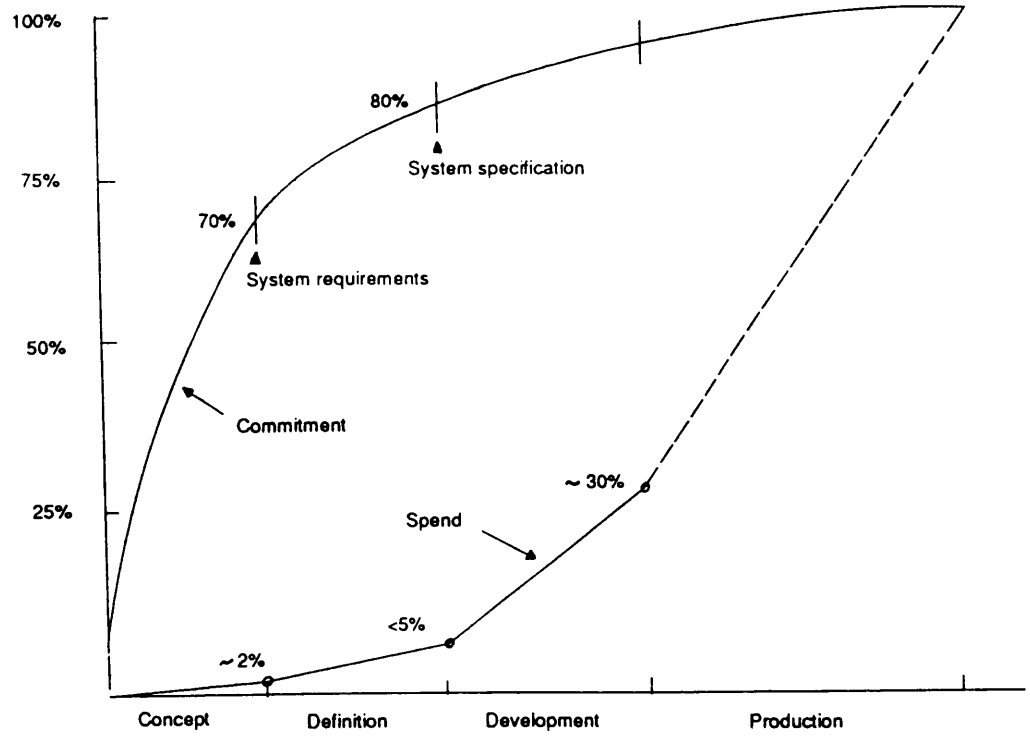


Figure 1.1 The cost spent and commitment, hardware and software [Win87]

If the trends in software costs are being also considered, then we realise that the 'software crisis' is important and is not just a cliche. Problems that have been addressed in this context are associated with schedule and cost estimates of software development which are often inaccurate. The software industry has often experienced inaccurate cost and schedule estimates as well as overruns in software development. Therefore, it is not at all surprising that meeting project deadlines has become a prime worry for managers in major industrial countries.[3]

The total world population of professional programmers in 1980 was estimated to be 3,250,000 [Jon83]; [Boe88a]. Of these about 1,000,000 were located in the US, 1,000,000 in Western Europe and 500,000 in the Far East (mainly Japan). The 1980 annual cost of the programming labour force in US accounted for approximately 40 billion dollars, which represented 2% of the GNP. With an estimated growth of 7% per year, the expected population and cost of programmers for 2000 is [Boe88a]:

* US 3 million professional programmers, at a cost of 400 billion dollars.

* World 10 million professional programmers, at a cost of 800 billion dollars.

By the turn of the century, the software industry is estimated to have total turnover cost of about 800 billion dollars world-wide.[4] Therefore, controlling and

---

3. Price Waterhouse International Computer Opinion Survey indicates that managers in US, UK, Australia and France considered meeting project deadlines as their major worry while, managers in Japan didn't considered this issue to be a problem [Pri88].

4. Total software cost trends, based on 1985 figures with a continued estimated growth of 12% per year (indicating a 5% annual increase in personnel costs and 7% increase in the number of personnel) [DOD85], [Boe87a], in billion dollars are:

	1985	1995	2000
US DOD	11	36	63
US Overall	70	225	400
World wide	140	450	800

saving even a small part of this cost is meaningful. The gap between the demand for new and upgraded systems and the ability of the software industry to fulfil this demand is wide. This stems from a shortage of talented software engineers, programmers and managers, which is the major cause for the increase in software cost.[5] However, this is only part of the problem. Analysis of a sample of nine US Federal projects [Usg79] (at the total cost of 6.8 million dollars) showed that less than a tenth of all projects are delivered on time and within the specified budget. Furthermore:

- Only 5% of the software had paid for been used ($0.3M), and less than 2% ($0.1M) had been used as delivered, i.e. without change.
- 29% of the contracted software was paid for but not delivered ($2M).
- 47% of the contracted software was delivered but never used ($3.2M) and 19% was abandoned or reworked ($1.3M).

Although this sample represents only a tiny part of total software cost in the US, it is well supported. This picture should alarm those who have the authority over software contracts and those who manage software development. What are the reasons for such a poor record? It is not a question of technological breakthroughs, for they are required on few, if any, software projects. So where does the problem lie? The actual difficulty has been in estimating the appropriate resources needed for the solution. This difficulty is derived mainly from the inability to identify and appreciate all aspects of the problem [Ton79].

Even so, this situation is only one side of the coin. There are other endeavours from which we can learn a lesson although a very different one. Mills [Mil80] of IBM, describes his experience with very large and complex projects

---

5. The world wide programming backlog for 1980 was estimated as 5 million programmers, which is 150% of the world wide calculated professional programmer for this year [Pri88]. This trend continued throughout the eighties. The backlog is defined as requests outstanding for new or improved computer systems. This is supported by the US Air Force survey which has identified a four years backlog of important data processing functions which cannot be implemented mainly due to limited availability of personnel [Boe88a].

which were delivered on time and within budget. One of the projects was developed for 8 different processors and involved 200 person years of effort over a period of 4 years. The successful development of this project and of a few additional projects, all of which were implemented using an incremental (evolutionary) strategy, is attributed to the strategy chosen for the development process. This approach enabled the completion and delivery on time and within budget for each of the 45 incremental deliveries. Mills's conclusion that *there were few late or overrun deliveries in that decade and none at all in the past few years"* [Mil80] contradicts the previous example. Though this does not represent a common view held by the professional and the academic communities, it does suggest a better way of estimating the software development effort.[6]

Computer hardware technology has advanced rapidly whilst customers and users' requirements have become more sophisticated. Software complexity has steadily increased, creating a significant gap between software technology and management. Software technology, which includes both managerial and development methodologies, was inadequate to satisfy the innovative users' requirements [Jen79a]. For example, only the convergence of computer hardware and communication enabled the introduction of office automation into organisations in an attempt to meet the long-time vision of a 'paperless society'. Although the technology exists, these expectations are still unfulfilled. Given the increasing dependence of organisations on software-based systems, it has become increasingly difficult and eventually will be impossible to return to the previous way of doing business, or even to continue effective operation if the computing systems are out of date [Leh89].

Increase in size of software systems, particularly when poorly structured, may cause the management effort to increase exponentially [Weg80a]. Unfortunately, software projects expand if not properly controlled. The evolutionary nature of system development is subject to a *"Law of increasing*

---

6. Fox [Fox82] supplies us with few examples of that type. Two major airlines, each sued its supplier because after $40 million, already spent, the system was not even close to working. A major European bank went to court for a $70 million claim over software.

unstructuredness" [Bel79b] unless specific resources are devoted to maintaining the structure during the system development and growth. Good management is essential to the development of successful and reliable systems. Software development requires a careful, intense management system, which must be aimed at ensuring the highest quality of delivered products within budget and constraints of the project. These two aspects of a project are interdependent. The quality management of the software development process cannot be implemented by ignoring its economic aspects. The ability to evaluate the costs and benefits of a potential project at an early stage is paramount.

1.6 SOFTWARE DEVELOPMENT AND EFFORT ESTIMATION

Estimating the effort needed for software development cannot be discussed without considering the software development life cycle (SDLC), a conceptual framework that underlies the development and management processes.

Software development can follow a number of alternative approaches to fulfil a desired set of requirements. Sequential software development where the development process is consists of discrete phases and stages (within the phases) implemented in definite sequence, or an iterative development process are two possible strategies. Customised software development, or modification of an application package are additional two alternative approaches. Each of these strategies can incorporate the use of tools such as, data-base management systems (DBMS) and report generators. Each strategy will make use of the universal view of software **development** as discussed below.

1.6.1 Software development and effort estimation as interactive and iterative processes

The software **development** process is composed of four principal phases, regardless of the specific development approach or the unique features of that project.

These phases are definition, design implementation and operation.

Figure 1.2 A general view of software development.

The main concern of the definition step is **what** should be done from the viewpoint of the users. In other words, what are the desired functions, what is the data to be processed (in order to implement these functions) and what is the desired performance of the new system. This is explored in detail in this thesis so that all implications are understood. The design phase addresses the questions of **how** the data is to be structured and how the overall hardware and software architecture are to be designed. In the implementation phase the designed architecture is constructed and the designed functions are implemented. The operation step focuses on **changes** to existing software. Changes that result from user requirements, from errors, from modification and adaptations due to changes in the technical, users and/or external environments.

But, project development does not occur in a vacuum. A software project is a part of a larger computer and societal system. Projects are being initialised either in long range planning for information systems, or in a Project Planning phase. Initialisation through a Project Planning phase occurs as a result of an urgent, or ad hoc need, where established priorities for development cannot be undertaken. Cost analysis for proposed projects is done in either of these project initialisation phases. Therefore, this thesis takes the view that the definition phase should be comprised of 2 phases, a Project Planning phase and a Preliminary System Design (PSD) phase. The concepts identified at the Project Planning phase are further analysed into system requirements at the Preliminary System Design. These are similarly transformed into detailed design that is then

implemented as a computer program. This is done at the Construction phase.[7] The processes of identifying what is desired and how it should be implemented are of an iterative and interactive nature. The process of estimating the software effort should be implemented in each of these major phases and should continue throughout the project life cycle, as shown in Figure 1.3.

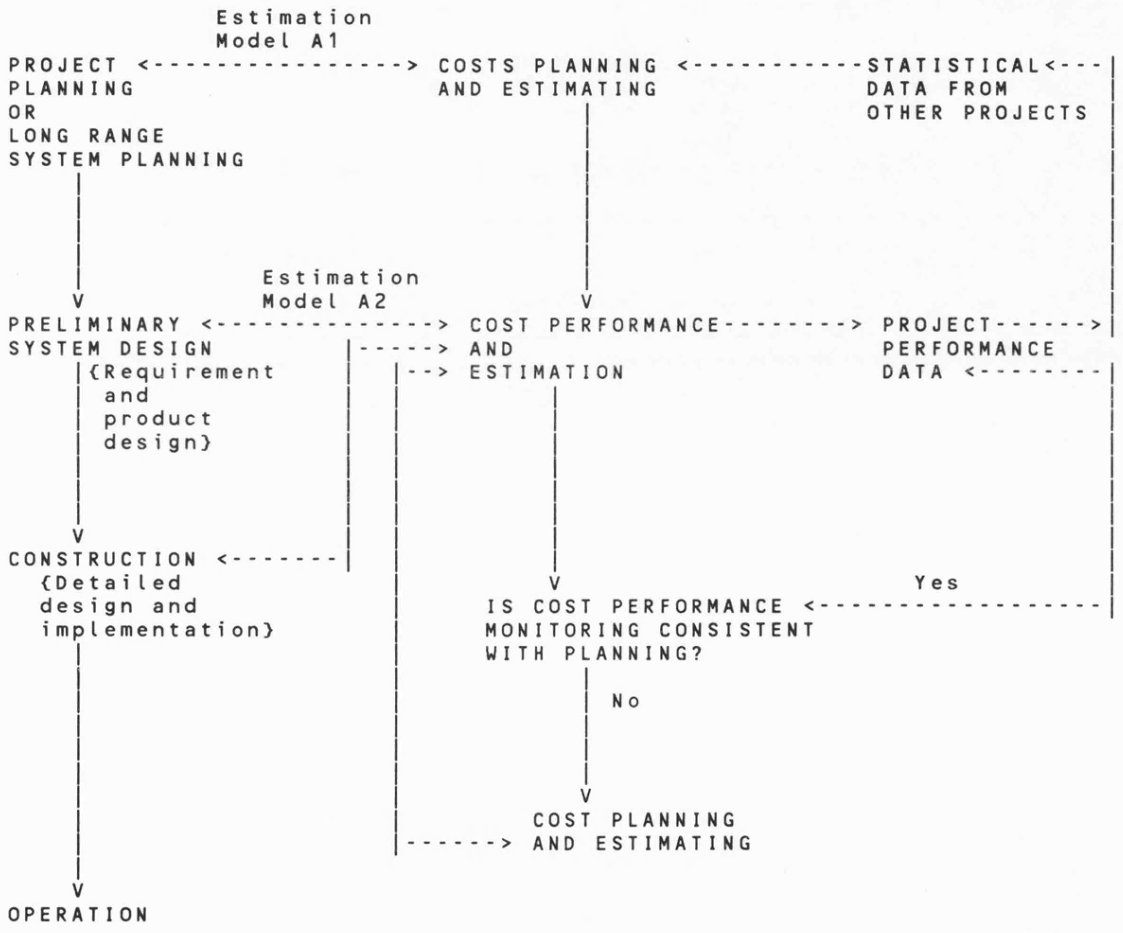


Figure 1.3 A general view, software development and effort estimating processes.

---

7. This phase is often called the development, production or implementation phase. However, as the main work of this phase is around the actual construction of the target system, the title Construction phase will be used in this thesis.

1.7 CURRENT RESEARCH

Research into effort estimation cannot be isolated from research within adjacent fields of interest. In order to understand the behaviour of the effort required for software development, researchers investigating it have approached the subject from different aspects. The earlier works of Benington, [Ben56] and Norden [Nor63] influenced Putnam [Put79], whilst Parr [Par80] proposed an alternative approach. Royce [Roy70], Boehm [Boe76] and Jensen [Jen84] modelled the process of software development, and concluded with various versions of a process model popularised as the 'Waterfall' model for SDLC. As new technologies became available, new paradigms for the SDLC were proposed and labelled with the generic term Prototyping.[8]

Researchers have long been interested in staff productivity using different software development processes and environments. The aim has been to define an agreeable and reliable measure of productivity for the software industry and perhaps to come up with a standard measure as is customary in the engineering arena. With such a measure we could record the effort of the SDLC (as a whole, or for various parts), measure productivity, establish productivity trends and hopefully predict the effort required for a new project.[9]

Researchers have been looking into the various aspects of the complexity associated with the software development process. Complexity is both a characteristic of the product to be developed and of the organisational environment. There are different causes for the complexity associated with software development. Complexity can stem from the behaviour of the people involved in the development, their attitudes, their roles in the organisation, their expertise and skills. A further aspect is technical complexity emerging from the development environment, such as new hardware and new development tools.

---

8. [Bal83]; [Ala84]; [Zav82;84]; [Agr86;86a].

9. [Wal77]; [Alb79]; [DeM85]; [Jon86]; [Dun83].

Complexity often results from uncertainty associated with the project and the development process. Entropy caused by interactions among system components (in the broad context of systems) and the feedback associated with these interactions, are the prime cause for the uncertainty which is inherent in the process of software development. The outset of the project life cycle is characterised by a high level of uncertainty.[10]

Empirical research in the subject of estimating software development costs has been done since the late 1960's in the area of estimating the effort and cost of software development.[11] In the early 1970's, with the dynamic growth of the information technology industry, the need to compete for large and risky software development contracts, illuminated the necessity to improve the process of estimating the effort needed to produce software. It was only in the late 1970's and the 1980's that intensive research resulted in a number of effort and cost estimation models, which were followed by the development of support tools for estimating the software development effort.[12]. However, a few new projects have been launched in recent years stemming from the work of the Esprit and Alvey committees, such as the MERMAID, SPEM, COSMOS and PIMS projects. Even with such intensive research no clear understanding of the behaviour of the software development process resulted.

Most of the models were developed, not coincidentally, by very large companies. These companies not only realised the need for estimating methods, models and tools, but they themselves could not afford to stay out of the race for software tenders and contracts. They needed support tools throughout the estimation process. This was the incentive to carry out intensive research into the area of software effort and cost estimation. But, the new approaches introduced by that work did not bring salvation to the industry. The early models were used by their developers (companies such as RCA, Boeing, Bell Laboratories, IBM) as

---

10. [MaC76;89]; [Gil77]; [Hal72;77]; [Che78]; [Cur81]; [Bas81]; [Bel81]; [DeM82]; [Leh89;89a]

11. [Nel66]; [Wol74;84]; [Dot77]; [Wal77]; [Hal77]; [Boe73]; [Put78;79;80;80a;81]; [Alb79;83]

12. [Put78;79;83;84;84a]; [Alb79]; [Boe81]; [Moh81]; [Jen83]

competitive weapons. The studies and research findings did not become 'common knowledge'.

1.8 DIFFICULTIES IN ESTIMATING SOFTWARE DEVELOPMENT EFFORT

The difficulties in estimating the costs of software development appear to stem from the management of complexity, in other words managing people and developing products in a dynamic and uncertain environment. Complexity is not a well-defined term in software engineering literature. A consensus does exist, however, as to its importance in estimating the effort for software development. The term appears in different contexts and is generally used in this problem domain, to indicate difficulty in the development process, caused by one or more factors. A definition which emphasises the interaction between all agents involved in the process was proposed by a panel which dealt with complexity in software development[13]:

> *Complexity is the measure of the resources expended by another system interacting with a piece of software. Categories of systems that may interact with software are machines, other software, people and even external environment* [Bas79a].

But can complexity, which is an intuitive concept, be quantified and measured?

The nature of the software product, its development process, the people who are involved in it and the environments in which the software is being developed and/or will be used, are important factors in the process of software development and that of software cost estimation. Understanding the characteristics of the software product and the factors influencing its development is essential.

---

13. After Curtis [Cur79]

1.8.1 The software product and its development process

The process of software development is characterised by the software product itself and the associated environments in which the development process takes place. During most of its development, software is basically an intangible product. Only as a working end product, which satisfies the requirements, is the software tangible and therefore, quantifiable. The fact that 50,000 lines of code (LOC) were written indicates nothing meaningful about the status of the development process. It can be used as a posterior measuring of productivity only when software is fully completed. On the other hand, the 'life-time' of this end-product is naturally limited as the application domain undergoes continuous changes. The software product should be treated as: *"an ever to be adapted organism rather than as a to be produced once artifact"* [Leh89]. This last point implies that a software product should be built in an evolutionary manner, to be modified and changed. The way to build software for that purpose is entirely different from development software as a 'black box' which is easy to use but need never be opened.

Desired software characteristics are often in conflict, requiring tradeoffs among factors (such as core storage requirement versus light code, flexibility or adjustibility to new needs, accuracy and reliability). The application developer of software finds himself in a dynamic environment. The software cost is derived by an extremely intricate environment in which many factors play a role and it is very difficult to isolate each of the factors influencing the software development. In addition, these factors behave differently in various situations. For example, a high response time might be of crucial importance for a project and, therefore, expands the effort needed for its development. The same factor might not be of importance in a different development environment and, hence, may affect differently the effort needed. Even within a given company, fluctuations in company project demands may dictate changes in priorities and thus affect project development dissimilarly. It is only the combination of all the factors of this environment which drive the cost of a software effort.

Volatility of the requirements. There is no one discrete correct solution to a set of requirements. Furthermore, there is no way to check the attributes of a

'correct' software solution, the same way in which the stress characteristics of materials might be used to check the properties of design for a building [Kit89]. Software requirements are often driven by forces far removed from the actual software laboratory. Product requirements in most cases are not well-defined nor are they frozen before any design activities began. Software engineers like to start quickly the detailed design and coding, they have problems stemming from frequently underestimating the time needed to understand the user functional requirements. This can result in software that does not meet the user requirements, which will inevitably alter during the lifetime of the project.

The control of the development process. There are virtually no objective standards of measure by which to evaluate the progress of software development. Software work is 'knowledge work' which can't be seen, does not fit into discrete tangible units and is difficult, if not impossible to measure. Therefore, a manager of 'knowledge work' faces difficulty in knowing when he has accomplished something. The abstract nature of software development makes it more difficult to manage. The physical visibility of a partially completed building must be replaced by documentation, that provides the state of a partially completed project, which aids in understanding and modifying the system.

Product development is often dependent upon the availability of supporting software, programs or data-base elements of another project. Delays in the availability of supporting programs or data-bases may subsequently induce slippages in the product development. The current practice, often used to describe the progress and the status of software development by indicating percentage of completion, is not satisfactory.

1.8.2 The people

Experience indicates that software size and complexity is generally underestimated [Boe81]; [Wol74]; [Bro75]. If we could understand the reasons for this phenomenon we might be able to overcome these obstacles. Cultural behaviour is considered as the cause. The people involved in software work are optimistic, they

desire to please and they tend to have incomplete recall of previous experience. Team members are generally not familiar with the entire software job. Furthermore, software engineers and programmers tend to 'gold-plate' the developed product to satisfy their own technical challenge aspirations rather than the essential functionality of the system.

Software engineers and programmers often have trouble communicating directly with users. They prefer interesting work which often gets done in preference to dull work, meaning that the latter is frequently postponed or ignored. The software manager often has more interest in the feasibility and technical issues of the project, while the business manager's awareness of the influence of the loose control, direct or indirect, is low. Both types of managers are not always motivated or equipped to consider an information technology project.

Lastly, managers of software development effort often find themselves in conflict and are forced to act defensively. They are asked to estimate the required effort and to take responsibility for the development process following their estimates. At the same time, they face pressure to lower their estimates, knowing that high estimates, although well justified, might not be accepted by higher authority, causing the postponement or even abandonment of the project.

1.9 THE ESTIMATOR'S DILEMMA

There is a need to know what the costs of developing a software product will be. Yet, it is difficult to estimate these costs accurately. It is even more difficult at the outset of the project life cycle. A strong link exists between the availability of estimates of effort and duration for software development, the functionality within the project problem domain being estimated and the capability to plan, manage and control the software development effort and cost. We cannot manage without being able to control and we cannot control without knowing what is to be controlled. In other words we cannot control without planning, nor can we plan without estimating.

The abstract nature of the software development process cannot be changed. Neither can we change the volatile nature of the requirements and the driving forces of this phenomenon. Yet, does the fact that it is difficult to estimate the effort required for software development imply that nothing should be done about estimating this effort? Does the fact that we cannot isolate the exact effects of each of the influencing factors on the software development process justify doing nothing about estimating this effort? These difficulties do not eliminate the need for effort estimates. Nor are they a reason to abandon software cost prediction and to improve methods of estimating.

> *Estimates deal with the unknown, and the unknown has a perverse way to subject poor developers to all kinds of rude shocks. I know only one thing that keeps these rude shocks to a minimum, and I shall take this opportunity to pass it to you: Good Luck!,* [DeM77].

1.10 LONG-TERM RESEARCH GOALS AND SPECIFIC OBJECTIVES

Perhaps there are avenues in which we could help the 'Goddess of Fortune'. We should not rest quietly waiting for Her to help us, although such help is truly needed. The subject of this research is about what should be done to help the software manager, the project team, the users and all who are involved in the process of software development with this matter.

Having discussed the difficulties stemming from the phenomenon described as the 'software crisis', recognising the trend of software costs, and having gained understanding of the difficulties associated with estimating the effort needed for software development, it is now the time to establish the long range goals and the specific objectives for this research. This research aims to pave the way for members of organisations who are parties to the software development and effort estimation processes, by proposing a concept which will aid the understanding of these complex processes. The long-term research goals are:

1. To describe a practical and systematic method of software estimation which

will serve as a guideline for the parties involved in estimating the software effort.

2. To identify the possibilities of automation and to specify a system which will make use of the 'common knowledge' which exists with regard to software development management.
3. To build an automated tool for future research.
4 To establish the foundation for an historical data-base for further research.

The problem of designing an effective model and tool for estimating the effort required for software development is both important and difficult. Yet, what can be done to help in solving the estimator's dilemma? The scope of this research is restricted to estimating the effort required during the Preliminary System Design (PSD) phase only. By so doing, a response will be given to a part of the process that needs special attention that it does not receive in current tools. This thesis offers a model which has the following advantageous properties:

1. A practical and systematic method of software estimation which would serve as guidance for the parties involved in estimating the software effort, and by that:

    * Assist experienced project managers and all other data processing professionals by suggesting an interactive and structured estimation process. This process which facilitates thinking about both their work and their decision making and allow the incorporation of the estimators' judgement into the estimation process.
    * Serve as a training tool for the inexperienced project manager and user, by proposing a standard procedure for software project development and for the estimation process.
    * Provide a basis for assessing project risk, comparing and evaluating the various development alternatives and for developing a working plan for a project.

2. A way to capture and to retrieve the assumptions underlying the estimation process and therefore to:

   * Keep the organisational knowledge and the knowledge associated with a project, and so they are not lost through personnel change.
   * Gain better knowledge and understanding of the process and the factors that influence it.

3. A software estimation process which is integrated into the process of software development.

These qualities will hopefully allow informal interaction among all parties in the development process: the user, the project manager, the project team and the organisation's management and hence, improve reliability among these parties, produce more precise estimates and decrease overruns.

1.11 RESEARCH DIRECTIONS

Background. The economic evaluation segments are the weak links in the various phases of the projects' life and they do not receive the degree of attention they deserve. There are many different reasons for this phenomenon. It is common to think that most projects get off on the wrong foot because the project definition and the project planning factors are not treated with sufficient attention and competence. The manageability of any development process is determined by the amount of uncertainty experienced during the development. Unfortunately these properties are particular to the Project Planning phase. Yet, the argument is that if the information system manager had an understandable, friendly and practical method, which emphasises the principles involved in the estimation process and the results, using tools of support, then he would have taken advantage of it and used it.

One way of supporting quality management in this area is to provide, at the

different phases of the life cycle, guidelines for the products to be supplied and the means for producing them. This will serve as a facilitator to the information technology manager and will assist in the dissemination and usage of the 'common knowledge' which already exists in the organisation, related to both the software development process and the effort estimation process.

One possible way to deal with the issue of estimating the cost for software development is derived from the world of engineering. In the software development process, we could employ a standard approach similar to that used for production planning and scheduling. Some qualitative attributes might be drawn from the production process. For instance, we might gain insight from analysing the software development process, the various procedures, their components, and the ways they interact and integrate, whilst bearing in mind an analogy derived from the engineering practice, the 'Bill of Material' and the 'Routeing' principles. This research will argue that it is feasible to approach software development estimates using surveyors, decomposing the software development life cycle into standards components which have an associated average effort needed to produce them.

Each of the SDLC phases employs activities characterised by various attributes. Therefore, this research takes the view that effort estimation for each phase should be dealt with differently. None of the current models, known to the author of this thesis, deal with estimating the effort for the Preliminary System Design phase (PSD) of software development explicitly.

Aiming to provide an insight into the process being modelled, this research advocates the bottom-up approach for estimating the effort for the subsequent phase (the PSD) and a top-down approach for the Construction phase. A better understanding of both the software development process and the effort estimation process will help in producing better estimates, since, the major obstacle is underestimating the effort which results from not knowing what is involved in a specific solution. The bottom-up approach assists in providing insight into the estimating process. This property is of high benefits although intangible.

1.11.1 The principles of the proposed solution

The Effort Estimation Model (EEM) developed in this thesis has the following **major principles:**

- **Life cycle.** Software development has a life cycle pattern which is composed of phases, segments and activities which represent the processes of transforming concepts and desires into a real operating software system. The concept is borrowed from the engineering world where the notion of product life cycle has long been used in product planning.
- **Phase - by - phase estimation.** The prime aim of this research is to develop a model appropriate for the process of effort estimation which takes place at the Project Planning (PP) phase and provides estimates for the **subsequent phase only.** At that phase the effort is estimated only for the PSD phase of the software development process. The effort needed for the rest of the project is extrapolated from these estimates using resource allocation among the development phases based on resource distribution among the project's phases, which is known statistically.
- **Activities.** Most activities involved in the software development life cycle have a standard list of cost drivers. The cost drivers serve as the basic unit for estimating the effort needed for each activity. A cost driver is, for example, an input document, a report, a file to be converted, a contract to be signed, etc.
- **Measurement.** Each of the standard list of cost drivers involved in system development has associated standards of effort. A Standard of Effort (SOE) is the amount of effort required to accomplish one work unit, or the amount of effort needed for a defined cost driver which is not expressed in work units e.g. system overhead. Standard of Effort (SOE) is the (organisational) inverse of a standard rate of productivity, measured in person-hours (PH). It is the result of measurement of projects performance, but with and heuristic adjustment process.
- **Judgement.** However, some of the activities involved in the software

development life cycle do not have a standard list of cost drivers and, therefore their standard of effort is not known *a priori*. The effort needed to implement these activities is estimated separately, by the effort estimator for each project, using his experience and expert judgement.

- **Complexity**. Every project has a complexity level which affects the productivity of a project. It should be noted and emphasised that the productivity rate of a project is a function of various attributes, among them staff ability and management competence which, also, inferred productivity. But, the aim of the estimation process at the outset of the project life cycle is to predict the person months (PM) required for the development. The 'natural' schedule, the number of various professionals and their required skills can be worked out only after the person months number, the effort required, is known and agreed upon. At that time it is not yet known who will be the individuals assigned to the project. Therefore, the only valid assumption about the productivity of a project, is a function of the general system complexity, uncertainty and difficulty associated with the system. Consequently, the project complexity affects the set of standards of effort associated with a project.

The concept proposed in the Effort Estimation Model (EEM) should be applicable for each of the alternative approaches for software building, however, with an adaptation to the specific approach.

The nature of the effort estimating task relies heavily on the judgement of experienced performers. Effort estimation of software development, in particular at the outset of the project, is an 'ill defined' problem and therefore, a closed algorithm is not an adequate solution for this process.

1.11.2 The Effort Estimation Model (EEM)

The EEM developed in this thesis assumes the use of a management framework for the software development life cycle (SDLC) of phases, which are composed of

segments (group of related activities) and activities.

A standard list of cost drivers is associated with each activity. The cost drivers serve as the basic unit for estimating tasks associated with each activity. A cost driver might be viewed as a further refinement of an activity and indeed, in some cases, the cost drivers are work components that identify the tasks to be performed. However, this is not always the case, some of the cost drivers identify an overhead for a system.

A 'standard of effort' (SOE) is associated with each of these combined entities, composed of a cost driver and a concurrent activity. The standard of effort associated with each of the cost drivers and corresponding activity may differ for an identical set (of cost driver and concurrent activity), according to the complexity of the project.

The proposition is to associate with each combination of activity and cost driver, three different 'standards of effort' according to the assumed complexity level of the system, complex, moderate or simple. The degree of complexity is considered as a subjective classification since human beings are involved in the development and in the complexity assessment. The various parties involved in the development may differ in their attitude and understanding of the project under discussion. The different groups might also have conflicting objective.

However, the standard of effort is not known for every activity involved in software development. There are activities which are characterised by a high variance of effort needed to accomplish them in different projects. Therefore, the estimates of the effort required to accomplish these activities is subject to the judgement of the estimator.

The Effort Estimation Model (EEM) is supported by a conceptual SDLC composed of phases, segments and activities, each activity is associated with one or more cost drivers which are correlated to a 'standard of effort'. The forecasted effort for some activities which differ widely in the effort required for their implementation is provided by the estimator based on his judgement, as schematically described bellow:

SDLC

Which consists of

PHASES

Which consist of

SEGMENTS

Which consist of

which have

ACTIVITIES <----------> COST DRIVERS

which consume

RESOURCES

which can be forecast based upon

MEASUREMENT and JUDGEMENT

which produces which leads to

STANDARD OF EFFORT PREDICTED EFFORT (PH)

which is measured in which is not a

PERSON HOURS (PH) STANDARD OF EFFORT

And is classified by the

GENERAL SYSTEM COMPLEXITY of the PROJECT

Figure 1.4 The conceptual view of the EEM.

1.12 RESEARCH METHOD AND THESIS STRUCTURE

The research incorporates the following **methods:**

1. Developing a conceptual model of the process of estimating the effort required for software development early in the life cycle of a project. engineering.
2. Acquiring knowledge, using walkthrough sessions, case studies and discussions, developing a questionnaire which imitates an estimation session.
3. Developing an algorithm for estimating the effort for the Preliminary System Design phase of the software development process.
4. Designing a support system and building a prototype using the suggested algorithm.
5. Capturing and analysing data, mainly for the purpose of tuning the algorithm, assessing the complexity rules incorporated in the EEM and evaluating the model.

1.12.1 Thesis structure and outlines

The following parts and chapters encompass this thesis.

Part I Focuses on two issues, setting the scene for this research and establishing the basic foundation for understanding the two processes of concern: the life cycle development process and the software estimation process. Two chapters are included:

Chapter 1: In this chapter the concept of software engineering and the estimator's dilemma were introduced. The need to estimate the required effort for the software development at the outset of the development process is contrasted against our inability to do so. This inability results from the complexity of the problem, uncertainty

related to the objectives and preferences of the people concerned, the lack of information associated with the complexity of the problem, and the uncertainty of the problem solving methods themselves.

Chapter 2: This chapter will focus on the processes of software development, and on the alternative methods for estimating this effort. Armed with the required understanding of the two processes and their implications on the desired structure of an effort estimation model, this chapter culminates with a short discussion of the primary concepts of the EEM proposed solution.

Part II These three chapters focus on the current research in the area of effort estimation which is a multi-disciplinary area.

Chapter 3: The ontology of effort estimation models, tools for software development and their evolutionary development is the subject of this chapter. Two models which represent the most commonly used approaches are discussed in detail. These are SLIM which represents the analytical approach and COCOMO which represents the composite approach. A comparison among the models, with particular emphasis on COCOMO and SLIM. The chapter concludes with a short summary of the current approaches and conclusions.

Chapter 4: Estimating the size of the software product is a prerequisite for estimating the effort required for its development. However, estimating the product size is a very difficult task which has implications for the ability to estimate the effort required for the development process. An error in size prediction results in a much higher error in the effort estimates. Two issues are addressed in this chapter, improving the size estimates and using alternative sizing methods which use a non-Line of Code (LOC) unit of measure.

Chapter 5: This chapter provided a critique of parametric models and complexity. Its starts with the presenting and analysing the results of

empirical comparative studies and the major findings from the discussion in chapters 2 and 3. Resource allocation among the phases of software development is of particular interest for this research and is analysed in detail. The discussion elaborates on themes associated with complexity of software development. They are: uncertainty, entropy and feedback as the causes of complexity. The major complexity determinants are analysed.

Part III Focuses on the Effort Estimation Model. The methodology used to develop the EEM and that which is incorporated in it are discussed. The structure of the EEM is described, accompanied examples of its use and a case study. Part III culminates with an evaluation of the EEM and with the contribution of this thesis to further research.

Chapter 6: The methodology used in developing the EEM is discussed.

Chapter 7: The Effort Estimation Model (EEM) is presented in this chapter. The discussion starts with analysis of the fundamentals of the EEM, followed by a thorough description of the model. A general system description, data models and functional chart of the process are given. The chapter closes with two case studies.

Chapter 8: The focus of this chapter is on the evaluation of the EEM. An evaluation of the qualitative feature of the EEM as presented in the prototype built, and a quantitative evaluation of the results from a field study are analysed.

Chapter 9: This chapter summarises the major issues addressed in this thesis, discusses the advantages of the model developed. The chapter closes with a look ahead, the contribution of this thesis to further research into the problem domain.

Chapter 2

SOFTWARE DEVELOPMENT AND THE EFFORT ESTIMATION PROCESSES

2.1 INTRODUCTION

This chapter addresses the processes of software building and estimating of effort required for software development. Various models used to direct and manage software development are analysed with the aim gaining of a better understanding of the process. The unique characteristics of each phase which together constitute the software development life cycle (SDLC), and its major concerns, will be considered. The nature of the estimation process and the classical approaches used in estimating effort will be introduced. The implications stemming from the analysis of the two processes will lead us to the basic assumptions upon which the Effort Estimation Model (EEM) will be based. This forms an introduction to the models and tools considered in the following chapters.

The processes of software development and effort estimation are strongly interconnected. Decisions taken at the outset of the software building process heavily influence the course of the development, its costs and schedule. Examples of such decisions include those associated with the approach chosen for the development process, the choice of support tools for this process and the decisions related to the functions incorporated into a software product and their desired quality. Decisions on software strategy clearly affect the effort estimates which, in

turn, affect decisions relating to the development process.

Software projects vary in many respects, but we can learn much from the similarities in the process of building software, which will be helpful in estimating the effort needed for developing software. These similarities serve as the infrastructure for software building. Therefore, the place to begin the analysis of effort estimation is with a study of the software development life cycle.

2.2 LIFE CYCLE MODELS FOR SOFTWARE DEVELOPMENT

The software development life cycle (SDLC) is essentially a heuristic process which serves as the basic framework for software development. The SDLC models are descriptive representations of the software process and the documentation required in each life cycle stage. The documentation, defined to be the satisfactory completion criteria for each stage, are the deliverables or the intermediate products of that stage. Even though the software development process is customarily characterised by a top-down approach and decomposed into stages, each having defined starting and ending points, it does not progress in a sequential manner from project inception to system implementation. The SDLC is an iterative and often evolutionary process. The primary functions of a software development model are to determine the order in which the major stages should be carried out and to establish transition criteria for moving from one step to the next [Boe88].

A number of software development models have been proposed: the conventional Waterfall model and its variations [Roy70]; [Boe76]; [Ton79]; the Iterative Enhancement to the Waterfall life cycle [Bas75]; the Canonical model [Leh84]; the Contractual model [Leh85]; the Spiral model [Boe88] and new paradigms such as the Prototyping [Sch83]; Operational Specification [Bal81;83] and the Transformational Implementation model [Bau82]; [Che81]. There are many representations of the life cycle, each subculture of the software industry has its own representation and each of these tends to be modified somewhat for specific projects.

2.2.1 The Waterfall Model

The traditional SDLC model, the Waterfall model, was foreseen in early works, such as Benington's [Ben56] who described a process model with the basic characteristic, subsequently termed the Waterfall model by Royce [Roy70]. However, Boehm's [Boe76] presentation of the Waterfall model, in which he described the model and its basic assumptions, was an influential milestone providing the economic rationale underlying the model. This model became the standard for software development in US government and industry.

The software development process consists of discrete phases decomposed into stages, implemented in a definite sequence, each of which aims to achieve a defined set of sub goals, before the next stage starts. These phases and stages although sequential are interdependent, and a change made in one may have significant influence on the other.

Four major phases are clearly identified in the process of software development. They are, the Project Planning, the Preliminary System Design and the Construction (including the: detailed design, coding and testing) and forming a complete software product life cycle model, and the Operation including maintenance.

The Project Planning phase

Project planning involves the development and the selection of the necessary course of action to achieve an objective. The Project Planning phase (PP) aims to develop an overall plan, a detailed programme for implementation of the plan and the method for controlling the progress, cost and time variables of the project. Project control consists of the appraisal of the performance and the execution of plans in accordance with the established standards, and the initiation of corrective action, if required. Thus, the availability of a plan and established standards of performance are prerequisites for controlling the effort required for project development. The main concerns of this phase are:

- To establish which resources are required for the development process, when are they required and to ensure that they are obtained.
- To evaluate the alternatives and to choose between them.
- To establish standards of performance and methods of control for the development process.

The Preliminary System Design phase

The Preliminary System Design phase (PSD) aims to specify **what** the system should do from the viewpoint of the user needs and the technical aspects to be solved, in order to develop the desired system's functionality.[1] Even though this phase is the most important of the development process, because it affects the rest of the process, it is the least studied and the least understood. The PSD phase is concerned with the problem formulation and analysis, the search for potential solutions, their evaluation and comparison. Three major stages are incorporated into this phase. They are system feasibility, system requirements and product design. The software requirements stage emphasises the user's view of the target system, while the product design emphasises the technical requirements. The product design stage deals with the functions needed to fulfil the users requirements and with the data necessary to support these functions. Various design alternatives are evaluated and iterated between the software designers and the users, until an acceptable design emerges which satisfies the user requirements.[2] The model assumes that all the required information about an application can be obtained prior to the development, and a concise and consistent specification of the proposed system can be produced prior to the product design

---

1. This phase is often called the requirements specification and product design phase. However, as it is in this phase that the preliminary architecture and the functionality of the target system is designed, it should be titled as the Preliminary System Design.

2. The process of choosing the preferred solution to the problem is tricky, it is usually based on trial and error, negotiation and social interaction.

stage.[3] Experience shows that various types of inherent uncertainties are associated with the software development process.

Formalisation of user requirements is a difficult task and is often ignored in favour of the easier one of developing solutions to what the programmer thinks the problem to be. The difficulty stems from the fact that users, more often than not, do not know how to state their needs in a manner that the software analyst can clearly understand. However, the main difficulty arises from the fact that the problem has been formalised by those who are not the problem owners. The term 'problem owner' is used to indicate the owner of the business problem that is the trigger for the target system. Communication of concepts between the users and the designers of the system, and later on in the process, between the designers and the implementors becomes a problem. This is especially true when a system includes the need for new hardware elements to be incorporated, particularly elements such as displays, logic chips and customised interfaces. The key considerations of this phase are in:

- Understanding the user requirements, mapping them onto a design which will eventually be approved by the users, and identifying the technical requirements needed to build the system.
- Setting the design baseline for the target system and ensuring that all parties responsible for using and operating the system, understand and agree with the key design and cost factors.
- Securing management commitment to the project and arranging for user participation in the development process.

There are good reasons for the identification of the Preliminary System Design as a critical phase. Firstly it is more costly to resolve software problems if they are identified further into the development cycle. Errors detected early in the life cycle can be solved much more easily and more cheaply than those discovered

---

3. This view point introduces problems and therefore invited critique from researches such as: Peters [Pet81], McCraken and Jeckson[McC81] and Swartout and Blazer [Swa82].

in later stages of SDLC. When using the Waterfall model, the first time a software project is totally assembled and tested is quite late in the life cycle, at the early part of the integration and implementation stages. Obviously, a great risk is involved in such an approach. Major implementation problems can only be tackled at a time when solving the problems would imply delay in schedule and overruns of costs. Secondly, the effort needed for software development accumulates exponentially, starting immediately after the PSD has been completed. At that transition point we are still in a position to terminate the project if necessary, having used only a relatively small amount of the planned resources and before a commitment is made for a high percentage of the total costs [Win87].

Analysis of system requirements provides a detailed foundation upon which the technical programmes and procedures will be developed. The initial emphasis is directed entirely to an analysis of the user operation. Once the user requirements and environment are understood, the technical approach is determined. The designers of the system then have a sound basis on which to proceed with implementation. The definition of the system is formally reviewed and agreed upon and the design baselines are revealed. Changes to the baselines are accepted and accomplished only through a formal change of control process.

The Construction phase

The Construction phase aims to specify the chosen solution in detail, to indicate **how** the requirements are to be met by the data processing system, and to construct the design of the system. The following major stages are included in the Construction phase: detailed design, coding, system integration and implementation. Each of these stages is accompanied by an iteration loop feeding back details to a previous stages or phases and thus forcing a more complete definition of requirements. This phase starts by further refining the definitions and system design which resulted in the design baselines. It continues with finalising the technical software and the system design necessary to complete the

software. The Construction phase focuses on completing, documenting and validating the design, establishing an approach for converting the system to be replaced, keeping the development status visible and controlled, establishing the testing approach; testing the system and delivering a quality system to the users. The major concerns of this phase are in:

- Controlling the construction process by monitoring project activities and progress. This is achieved by maintaining the design at a proper level of refinement, based on the design baselines and by keeping programming simple. Unless requirements for response time or other constraints make it absolutely essential, unnecessarily complex programs should be avoided.
- Avoiding the 'after-thought' and the 'gold-plating' syndromes. The temptation to incorporate changes which are improvements or modifications to the design baselines often exists. Such refinements should be employed only after thorough consideration of costs and schedule delays that may be incurred.
- Correcting errors as soon as detected. As time passes, designers tend to forget the reasoning and rationale used to support an approach or a particular technique. Therefore, the early detection and correction of errors is of great value to the development process.

The Operational phase

Finally, when the system test is formally completed and audited, the software is transferred to the user's control. The Operational phase starts at that point in a project and is not part of the 'development effort', even though the target system may undergo changes. This phase embraces all the activities that are required to continue operational use of the software. The required modifications resulting either from errors discovered while software is operational or from the need for software upgrades, are accumulated and evaluated periodically. The evaluation process aims to establish short-term and long-term strategies for the employment

of the desired changes. Each change, if major, should be considered as a new project or an additional increment to the system. The formality of the SDLC is again employed, either in total or in part. The main considerations of the Operation phase are:

- Identifying and controlling the cost, schedule and sequences of the desired changes to the system.
- Assessing the quality and structure of the system to provide a basis for future planning.

2.2.2 The Verification and Validation (V&V) concept

The ability to minimise the risk associated with the development process, and the power to control the actual development process, are critical to the success of software development and in particular to the PSD phase. Experience has shown that the most extensive cause of late delivery of software and inadequate performance is an ineffective requirements analysis [Deu79]; [Boe81]; [Fox82]. The introduction of the Verification and Validation (V&V) concept aims to improve the means to deal with these key issues. A constant iteration takes place between levels, as analysis and synthesis at one level uncover deficiencies in the design at a earlier level. Similarly, an iteration loop takes place between the stages of the life cycle, feeding back to the predecessor stage and thus forcing a more complete definition of requirements as illustrated in Figure 2.1

The verification process aims to assure that the right product is being developed that each level of requirement or specification correctly echoes the desired requirements. The validation process aims to assure that the right product is being built, that each end item functions and contains the features prescribed by its requirements or specifications. The V&V processes are addressed in each stage of the SDLC and are the major means of providing quality assurance to a software system. They are often referred to as Configuration Management. This formal mechanism minimises the expensive rework involved in feedback across

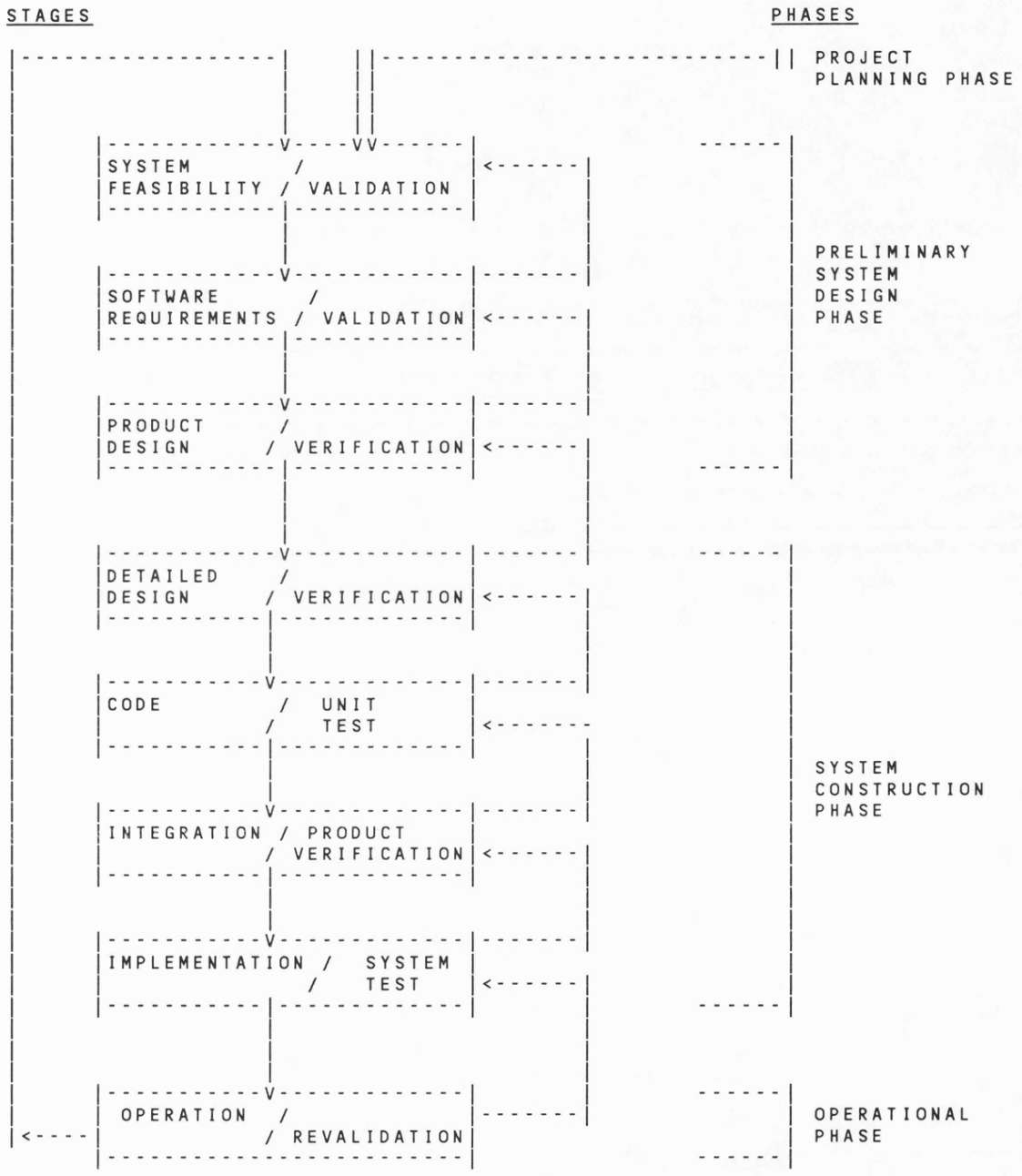


Figure 2.1 The Waterfall model including the V&V process [After Boe81]

many stages, since a major source of reworking results from misinterpretation of requirements.[4] Each stage culminates with a verification or validation feedback loop to the predecessor stage.

2.2.3 Deviations from the Waterfall route

Presumptions implicit in software development following the Waterfall process are that the developer knows the requirements, that the requirements are stable and that an efficient approach to satisfy them can be designed. However, the real life situation is very different, in particular when systems are either new to the user and/or to the developer, or when the subject of development is state-of-the-art. The sequential approach, usually forced by the Waterfall model, is not appropriate for the development of software in situations where either the user is unable to define the nature of the system to be solved, or where there is no simple solution to the problem. In such circumstances, the problem description and definition can benefit from extended exposure of the user to the data processing capabilities in the real environment. Thus, only when the user gets the first version of a system can he recognise the capabilities of the technology and what it can supply. This is achieved by either of the following approaches: 'throw it away' prototyping or incremental development within the framework of the waterfall approach.

The initial incorporation of prototyping via a 'throw it away' or 'build it twice' step, helps the user to accumulate experience with the functionality of the target system and its behaviour. The user is then more capable of adding to and changing the original requirements. The prototype is implemented in parallel with the requirements analysis and product design [Boe81]. Only mainline functions,

---

4. An often quoted failure, that perhaps could have been avoided, if the initial requirements and specification had been validated and verified, is described in the US Congress Bulletin [Con76]. The initial requirement for the Advanced Logistics System, contracted by the US Air Force, was that 90% of the transactions should be performed online. However, previous to cancellation of the project it was quite clear that only 10% of the transactions needed to be performed online. The US Air Force spent over $300 million in a futile attempt to automate this system [Boe81]; [Fox82].

which are related to the basic tasks desired by the user, should be implemented on a trial basis.

The second deviation from the sequential development is employing the incremental strategy. Recognising the difficulty of achieving a good design for a new system on the first attempt, Basili and Turner [Bas75] suggest incremental development as a way to cope with the uncertainty associated with the specification of complex systems. This approach forces top-down implementation to be incremental, and the increments represent all the functions desired for the target system. However, a parallel development of increments can start only when the Preliminary System Design is completed and verified. Thus, taking this approach forces solid preliminary design work and a careful selection of the appropriate system for incremental development. Each increment is developed and delivered to the user. The lessons learnt from the development are incorporated, if applicable, to the increments not yet delivered. The aggregation of the increments become the total target system. This approach allows for the implementation of the design-to-cost approach discussed in Section 1.4. However, taking this approach may reduced the alternative solutions.

The advantage of the incremental strategy over the prototype 'build-it twice' approach, stems from the difference between the process of building a total system to that of building successive increments. In the traditional development process a 'prototype' is produced by using iteration over the entire development cycle. Thus, the effort accumulation curve needed to build the system behaves differently in each of these approaches. When the prototyping strategy is taken, the requirements are completed only after the prototype has been built, exercised and approved by the user. Therefore, the incremental approach provides a less expensive way to incorporate the users' experience into a refined system than the total development involved in the 'build it twice' approach. The effort accumulation curve takes a 'flatter shape' when developing in increments than when the prototype approach is followed.

2.2.4 Motivation for the new paradigms

The key factor in the search for alternative methods for developing software was the convergence of the recognition that Waterfall model has not been able to satisfy completely the requirements evolving, with a dramatic change in the state-of-the-art of the industry. The cost of hardware has been declining significantly, while software costs have been increasing constantly since the introduction of the Waterfall model in early 1970's [Cor80]; [Boe81]; [Mus83]. Computerised information systems, often complex, incorporating distributed and communication capabilities, became essential to the operation of many sectors of society. However, the demand for new and updated systems is not being met as result of an extreme shortage of adequately trained professionals, a shortage which causes manpower to be a very expensive commodity in the process of software development. A need therefore, has developed to accelerate the software building process.

All critique of the Waterfall model is centred on its inflexibility. The model fails to provide adequate mechanisms both for managing the inevitable changes in requirements and for involving end users throughout the development process. Although the Waterfall model employs a systematic approach to software building, in which a successful system is achieved by attaining sub-goals in a particular order, it does not provide much insight into the processes occurring within these phases. In addition, the emphasis on fully elaborated documents as completion criteria for the requirements design is a primary difficulty and not always effective. Organisations undergo changes continuously, therefore, when the specified system is completed and delivered to the user, it is often no longer desired. Curtis [Cur87] sees that the major shortcoming of the model is in its failure to treat software building as a problem solving process:

> *Not only the developer trying to solve problem presented by stated requirements and the constraints of available technology, but customers are also trying to solve a problem for which they believe the requirements will yield a solution. Yet, since customers often don't understand the subtleties of their problem, and even more often don't understand the limit of technology, software development becomes a problem solving process involving multiple agents.*

An additional critique of the Waterfall model arises from the fact that it is difficult to extricate the analysis of what should be done, from the synthesis and how it should be done. For many years, researchers advocated the need to separate specification from implementation.[5] This rests on the assumption that the analyst has, or can obtain, a detailed understanding of the problem, can implement a solution and move on to another project, leaving the maintenance to others. But, since the new hardware and software technologies afford us more flexibility in software building, this separation now seems artificial. Too many problem-oriented issues, such as decomposing high-level functions, and system performance constraints, are left to impinge on design decisions.

Similarly, researchers who advocate the evolutionary approach to software building, such as Lehman [Leh84;85;85a;87], Dixon [Dix88] and Williams [Wil88], are opposed to the separation of the Construction and the Operation phases. They argue that software development is not a 'one-off' production process, followed by maintenance. They believe instead that it is an evolutionary process throughout the system life cycle. Upgrades and changes are constantly needed. This is the philosophy which led to the development of the Transformational Implementation paradigm, which is discussed in Paragraph 2.2.5.

2.2.5 The new paradigms

The search for alternative methods of specifying requirements motivated the new developments. Users and developers of software systems felt that *"it is really impossible for a client, even working with a software engineer, to specify completely, precisely, and correctly the exact requirements of a modern software product before trying some versions of the product"* [Bro87]. The earlier an activity occurs in the SDLC, the less we understand about the nature of the activity. Therefore, alternative ways to facilitate the process of understanding user requirements and hence speed up the software development, are of great benefit. The new

---

5. [Knu74]; [Dij75]; [Mil80];[Bau82]; [Sch83]; [Bal82;83]

paradigms for system development exploit the advances in computer technology itself, through powerful and high-level software tools, made practical by inexpensive hardware. These technologies provide us with the capability to develop a quick version of the software system or part of it, which can then be evaluated and re-specified if necessary. This is known as prototyping.

The process of prototyping aims to clarify the characteristics and operation of a system or a part of it. It is used for exploration and experimentation, which often takes place with the user in an operational environment and before the 'real' system is developed. Issues which are difficult to specify, using the traditional process, are often addressed, e.g. user requirements, user interface, feasibility design and system performance. The process is a **continuous** one, until the fit between user and system is acceptable.

The terms 'prototyping' and/or 'rapid prototyping' are frequently used as generic terms for all models of the new paradigms, often classified as Prototyping, Operational Specification and Transformational Specification.[6] These three models are partial models only, in the sense that each of them responds to a different need or disadvantage experienced in the conventional life cycle model. The information gained from the implementation of a prototype in understanding the users needs, the operational and the design feasibility, can be incorporated into the conventional life cycle procedures, and thus improve both the complex communication and the feasibility decisions involved in the process.

Prototyping

As noted above, the most cited of the new paradigms is the prototyping model, the building of an early version of a system or a part of it. A software prototype mainly aims towards producing rapidly and cheaply, as early as possible, a working

---

6. Other terminologies and classifications for the various types of prototypes exist. Floyd [Flo84] suggests: Exploratory, Experimental and Evolutionary. Law [Law85] adds the Performance and the Organisational prototypes, which are special cases of the Experimental prototype. Yet, the Organisational prototype aims to evaluate the associated organisational implications.

model of a system, and gaining information about the problem from it. This information is used later in the development process of the operational version of the software product. This type of prototyping is often referred to as Specification Prototyping [Keu82], Specification by Example [Chr84] or Exploratory Prototyping [Flo84]; [Law85]. The prime value of the prototyping approach is in the PSD phase, particularly in the product design stage, for the purpose of feasibility evaluation.

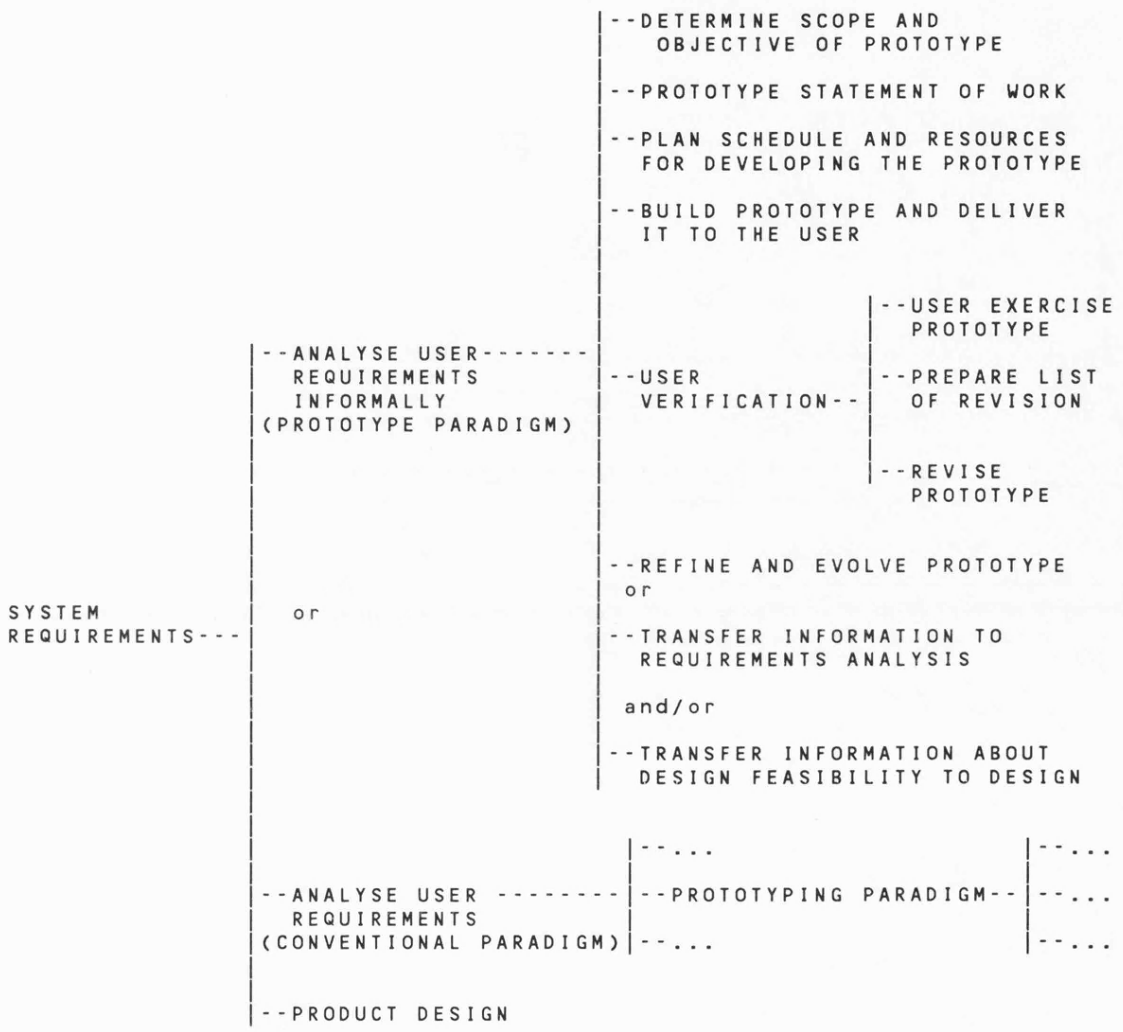
Two schools of thought exist. The first advocates the use of a prototype approach only as an initial version of the system, which is thrown away when the needed information is gained [Geh82]; [Bud84]. The second suggests that a prototype may become the final version of a system by means of an evolutionary development process. In cases where a working prototype is identified to provide a core of functionality certified by the user to meet his needs, it is feasible to extend this core into a final system. This approach was suggested as early as 1975 by Basili [Bas75] as a alternative route to the Waterfall model. The approach is highly recommended by researchers as: Mills [Mil80]; Scharer [Sch83] and Gilb [Gil87]. However, only the new technological capabilities such as 4GL'S and 'formal specification' methods, justify them being called new paradigms.

The prototype approach can be implemented either as a separate route, or accommodated within the traditional Waterfall SDLC. The information gained about the user requirements or design issues is transferred to the relevant processes in the conventional SDLC, to be incorporated into the final requirements and design specification. See Figure 2.2.

By adopting the prototype approach as part of the conventional SDLC, we may gain:

- Improvement in the communication between the various groups involved in building the software, mainly information technology personnel and users. This is achieved by relating the prototype to user experience.
- Simplification of the process of identification of the user's real needs. This can be achieved since the flexibility to adopt changes, in the perception of user's needs, is relatively easy when using prototyping tools.

PRELIMINARY SYSTEM DESIGN



SYSTEM CONSTRUCTION

Figure 2.2 The prototype paradigm and its relationship to the conventional SDLC.

- Efficiency in understanding the system characteristics. As a result, all parties that have participated in the development process are better armed to evaluate high risk issues at an early stage of the development process and possibly to avoid them. The use of this approach gives us an additional opportunity to alternate strategy or course of action, when it is still possible, at relatively low cost and risk.
- Assistance in the design and the operational feasibility processes, hence reducing the 'deadline' effect of the project. This can be achieved since an initial version of the system is delivered and available to the user early in the SDLC.

Discussion

Prototyping is not the panacea to all software problems. There are some crucial issues associated with software development for which the prototyping approach does not provide information. They are discussed below.

Effort required for the implementation of the final system. It is generally possible to obtain a large portion of the most valuable capabilities of a system after implementing only a small part of that system. However, the effort consumed for building a prototype cannot be extrapolated into the effort requirements for the total system. Building the total system will differ from the prototyping in the use of tools for the development. Tools such as 4GL's and small Data-base Management System (DBMS) packages are generally used for prototyping. However, the real system will often reside on the organisation's DBMS which differs from the experimental one used for prototyping.

System interaction with other elements in the software environment. Figuring how to handle interfaces between elements of the systems is a crucial issue in any software development, and if not well planned it will cause significant difficulties mainly in the systems integration stage.

The major aim of prototyping is to gain information about user requirements and feasibility issues, which are otherwise difficult to visualise. Since

major decisions are based on the results of prototyping, the value of this information lies in getting it as early as possible in the development process. Otherwise, the total planning of the system suffers. System planning, more often than not, is done less comprehensively when prototyping than when specifying a system. One of the neglected areas, resulting from this phenomenon, is the systems interface.

System behaviour in extreme situations. When creating a prototype performance characteristics such as speed, security, accuracy or completeness of error handling are sometimes compromised. What is achieved from prototyping may be enough for the users to extrapolate what they want, whether the requirements were correctly understood by the implementors or whether it is feasible to implement them as requested. But, to get the full performance of the system may require much additional work.

The Operational Specification

An Operational Specification is a prototype, sometimes called a functional prototype. The basic idea of this form of prototype is that a system can be specified using a formal specification language, that has a precise meaning and therefore can be executed directly. The paradigm has a twofold aim: exploration of the behavioural aspect of the system and improvement of maintainability. Operational Specifications are machine processes, written in a language which is not understood by end users nor other non-technical people. Computer specialists specify the desired system, in terms of implementation-independent structures, that generate the behaviour of the target system [Bal82]; [Zav84].[7] Thus, the Operational Specification output can be seen and evaluated by the end user. When using the traditional development process, the functional behaviour of a system can be analysed, only very late in the software life cycle, after the

---

7. The structures provided are independent of a specific hardware or software configuration, while the conventional design process refers to a specific environment.

implementation stage. But, when implementing an Operational Specification process it is possible to evaluate and interpret the specification, to show the system behaviour and hence to help in various feasibility decisions. By using the Operational Specification approach we may gain:

- A shortcut towards the understanding of the functional behaviour of a system but not the efficiency aspect of the system[8]. This can be achieved as the communication process between the users and the developers about the preferred behaviour of a system benefits from implementation of the Operational Specification, for the purpose of approval, even though it is not readable by users.
- A basis for rapid prototyping. Since by implementing and evaluating an Operational Specification prototype, the essential relations among the system elements are captured, it can provide a basis for interpreting the specification.

The Operational Specification responds to the problem of separation between **what** should be included in a system and **how** it should be built, as discussed in Paragraph 2.2.4. However, by so doing the paradigm introduces new problems of over-constraining and premature Operational Specification. Formal specification responds to significant decisions early in the SDLC, yet these decisions cannot be validated until the very end of the development process. The Operational Specification prototype often introduces internal structure (detail design function) into the process before specifying it thoroughly.

The paradigm aims mainly to enhance maintainability. However, the price is a less efficient system. Therefore, after the specification is stabilised, it is usually compiled. But, if compiled then it contradicts the evolutionary development objective. Although the Operational Specification process consumes more effort

---

8. Efficiency under extreme workload is very difficult to predict. Efficiency of workload is evaluated by a prototype running in an operational situation with simulated workload. But, as a prototype is only a simplified version of the target system this aim is difficult to achieve.

than the conventional specification, part of the output can be re-used in other Operational Specification processes. This approach can be employed as an integral part of the conventional life cycle, or only for a demonstration. Similarly, in addition to the formal specification resulting from the implementation of this process, the Operational Specification can be used as an input to the Transformational Implementation process.

Transformational Implementation

The Transformational Implementation is an approach to software development that uses automated support to convert a desired specification, which is written in a formal language, into a concrete software system.[9] This paradigm aims to reduce the labour intensive aspect of software development. It responds to the constant needs for safe, verified and reliable software[10], as well as for upgrades and changes to working systems. These needs cannot always be postponed until a periodical change of control process takes place. By using the Transformational Implementation, the separation between the development and the system maintenance is not so sharp as employed traditionally.

The process starts with a formal statement of a problem or its solution and ends with an executed program. The formal specification of the desired system is automatically transformed into system design and code. Successive application of transformation rules that preserve corrections is constructed and iterated to optimise the results. Most methodologies of this class consider the relationship between data and processes. The functional specification is prepared as structured outlines of the two objects: processes and data aggregates.[11] Three products

---

9. [Bal81]; [Che81]; [Bau82]; [Agr86].

10. These objective are of special importance when dealing with production of correct chips, critical software for weapons or nuclear systems.

11. The analyst is free to start defining either process or data groups since there is no algorithms which will produce repeatable specifications as the link between the data aggregates and the processes.

emerge from the process: a formal specification; a delivered system and a formal record of the sequence of the transformations, and the decisions taken during the process. The expected benefits from this model are:

- Reduction of labour intensiveness.
- Preservation of correctness, as a result of applying automated tools for transformation.
- Elimination of final product test. By applying such a process it is guaranteed that the final version of a program will satisfy the initial specification.

The Transformational Implementation paradigm can be used in a wide scope of application such as general support for program modification [Par83]. This may include the optimisation of the control structure, efficient implementation of data structure or rule generation. Additional applications are program synthesis, program adaptation to a particular environment, or program description by building a family tree of algorithms. It is essential, therefore, that the user of this approach is armed with a thorough understanding of the technical details of the system such as I/O, internal representation, mode of operation and a understanding of the implementation technique.

Two points of weakness are indicated [Zav84];[Blu84]. The first one relates to the maturity of the model which they considered an 'un-developed' methodology. The second has to do with the difficulty associated with the management of these processes. Managers may find it difficult to guide this process and analysers find it difficult to analyse.

2.3 RESEARCH FINDINGS

Boehm [Boe84] describes a case study in which seven teams developed the same small application: 4 of the teams used the traditional method of software building and 3 of the teams followed the prototyping approach.

- The prototype products averaged about 40% smaller, in Lines of Code (LOC), than the specified products and they required about 45% less effort.
- The effort needed to implement a prototype did not tend to produce a higher productivity level in the common productivity measurement, lines of code per person-month (LOC/PM). However, the prototyping approach produced more user specification per person-months than the conventional development.
- The prototype product led to somewhat better maintainability.[12]

In comparison with the conventional SDLC, however, the prototype paradigm tended to create several negative effects:

- Prototype implementations consumed proportionally less effort for planning and designing and more for testing and fixing the system.
- System integration appeared to be more difficult in prototype implementations due to lack of an interface specification.
- System design phase appeared to be less coherent in implementing the prototype approach than in a specified product.

2.4 THE TRADITIONAL & NEW PARADIGMS FOR SDLC - DISCUSSION

The reader is now acquainted with various approaches currently in use for managing the process of building software, for which estimates of effort are required. Two paradigms, very different in their basic assumptions, were presented: the traditional Waterfall model and the new approaches for prototyping and Transformational Implementation. The first one, the Waterfall model represents a conservative alternative which misses the flexibility needed to support a dynamic environment. The inability to present the user with a product

---

12. The maintainability was evaluated subjectively by asking the students who participated in the experiment to grade their preference in an ordinal way.

other than a verbal or graphic description is a real obstacle. It is difficult either to identify the real business problems, or to visualise the technological opportunities which can be incorporated in a system to satisfy these needs. The second alternative presents a flexible approach which overcomes this obstacle. The new paradigms enable us to demonstrate the system or part of it early in the SDLC, and to exercise the use of the product and its capabilities in a semi-operational environment. They enable us to conclude if this is really the expected product or if there is a need for revision. Or, if it is desirable and possible to extend the capabilities of the demonstrated product, in an evolutionary process, so it will become the target system.

However, the new paradigms are not a homogeneous group of models. The first representative of the new paradigm, the prototype, aims mainly towards exploration and experimentation of user requirements. The implementation uses 4GL tools. The Operational Specification is also a prototype and has the same aims, mainly towards the experimentation part of the system behaviour. Implementation of the Operational Specification involves the use of a formal specification language. Its output can be used as the input for the process of Transformational Implementation. But, a conflict arises when the output of the Operational Specification is transferred to the process of Transformational Implementation. The objectives of the two models differ and thus both cannot be optimally achieved. The main objective of the first paradigm is maintainability of the software, while the Transformational Implementation paradigm aims at optimisation of the performance.

The Transformational Implementation represents a totally different approach, will involves automatic transformation of programs and systems. The Transformation process might be from one language to another, or transformation of a system by automating the selection of transformation rules and optimising the process of system execution.

Each of the new paradigms responds to a particular problem in software development. Therefore, they are mainly partial models which can be integrated into the various stages of the software development life cycle and hence, improve the conventional process. Yet, these models can be used as stand-alone models.

However, they do not offer comprehensive solutions to the chronic problems, rather only modest improvements in productivity. Productivity and quality can be achieved from employing the new models, but only through skilful management.

2.4.1 Issues requiring special attention when using the New Paradigms

The new paradigms for software development require careful attention to economic considerations, to management implications, to potential pitfalls, and to the question of when to use them.

Economic considerations. The economic rationale of the new paradigms is based on the tradeoffs between hardware, software resources and scarce skilled manpower, since the industry is no longer hardware bound, but instead, limited by the number of experienced people. Prototyping has often been dismissed as a practical approach because it is considered expensive. Certainly, the costs of building a prototype are influenced by the availability of appropriate development tools, which are quite expensive if their cost is applied to one project only. But, if these costs are considered in terms of the organisational overheads and applied to a number of projects, a different picture emerges. Indeed, Gomaa and Scott [Gom81;83] and Zelkovitz [Zel80;82] reject the notion that prototyping is too expensive. They argue that although somewhat higher costs are agreed for building a system using prototyping, the aid offered by the approach is of high value to the requirements stage.

Implementing a prototype involves the expensive time of the most knowledgeable people in the users' organisation. Time cannot always be secured for this process, even if planned for ahead of time.

A crucial aspect associated with prototyping, and which has an economic effect on the development process, is setting the scope for a prototype and deciding when to stop the iteration process for revision purposes. These decisions heavily influence both the effort needed for prototyping a system, and the ability to estimate this effort. The only possible way to keep the development process under control and effective is if the implementation rules are defined at the outset of the

process.

When to use prototyping. The prototyping strategy should be used, where a learning process is involved in articulating either the user needs or the technical means required (e.g. an algorithm) to build a system with the desired properties. The Prototype and the Operational Specification models serve as a mechanism for exploration and experimentation in order to identify and to clarify the user requirements and to support the decision process of what is technically feasible. The flexible development process, in which the role of automation is increased, allows us to deliver to the user an executable object in the early stages of the software building. Therefore, the prototyping approach plays a great role in reducing uncertainty and risk associated with software development. The risk anticipated from user abandonment of a system is largely reduced. Hence, the tangible contribution unfolding from the new paradigms is mainly at the early SDLC stages and provides a risk reduction capability.

Management aspects. The management process of the new paradigms is more complex than that which controls the conventional Waterfall model. The flexible development process requires more management effort and is of a different style, as more uncertainty and changes are involved in the process. The users are heavily involved in this process, which aims to better their understanding of the system requirements and its behaviour. The emphasis, while employing these strategies, is on a fast response to requests for changes, using high-level hardware and software tools. The use of a formal mechanism, for control of change for each modification required, is therefore not appropriate as it slows down the development process. Yet, although not easy to manage, building a prototype should not be exempted from the management process by the reason of it being a 'quick and dirty' product which is not incorporated in the released version. The basic conventional sequence of activities: specification, design, coding and testing, should be maintained. Documentation of the positive and negative lessons learned from each implementation of a prototype is of great importance. Thus the new paradigms require a different management style, which emphasises the control of an environment which changes often and provides the tools to match it.

Potential pitfalls. The dominant purpose of the new paradigms is to buy information and they should be used for this purpose only, unless otherwise planned at the outset of the project. When a new system concept or a new technology is used, the 'throw it away' approach is beneficial since, even the best planning is not so omniscient as to get it right the first time. Therefore, the management dilemma is not whether to built a pilot system and throw it away, but to plan for it in advance. A potential danger involves incorporating prototypes in a released version without designing them to do so.

State of maturity. The Operational Specification and the Transformational Implementation models let us formally specify an idea which can then be transferred to a concrete system. The concept employed in these two paradigms is very promising for software development and for the development of process models for software building. Although organisations, very advanced in information technology, have been incorporating these tools, in their software development process since the beginning of the 1980's, the Operational Specification and, in particular the Transformational Implementation are as yet in an immature stage for commercial use.

2.4.2 Summary of discussion

Prototyping as a paradigm, like the classic life cycle, can be problematic, mainly when the incremental evolution of the prototype system is integrated into the final system. Here, the overall software quality is not always considered. Choices made for demonstration purposes and thus not always the best for real systems, become an integral part of the system. Long term maintainability suffers. A failure to develop an overall system plan before prototyping individual modules, can cause system integration to suffer. The new paradigms require a different management style, which emphasises the control of a changing environment and must provide tools for solution. Although the opportunities resulting from the new paradigms are of great value to the process of software development, they introduce a new set of problems. The aim of the Prototyping and the Operational Specification is

mainly to demonstrate a working product early in the software life cycle. But, the overall planning of the process often suffers. Thus, integration and interface problems arise later on in the process, when correcting them becomes a complicated and costly issue. The Operational Specification often introduces the problem of premature specification and, when integrated into the Transformational Implementation process, a question of efficiency versus maintainability arises. The Operational Specification aims to optimise maintainability while the Transformational Implementation attempts to optimise efficiency.

However, there is no general agreement as to what exactly Prototyping, Operational Specification and Transformational Implementation should mean within the context of software engineering. The new paradigms, mainly the Transformational Implementation, are as yet in experimental stages, not all aspects of their implementation are clear. Furthermore, even the classification often used in the literature and discussed in this chapter is not the only one that exists.

2.5 THE PROCESS OF EFFORT ESTIMATION

Having analysed the processes of software development, it now becomes obvious that the iterative nature of the software development process is a dominant characteristic. The process of estimating the effort for software development should take the same form. Effort estimation should thus be integrated into each of the SDLC phases, starting with high level estimates at the early life cycle phases. These estimates are further refined and updated when new data is available, when uncertainties associated with project functionality are reduced and when the level of complexity is understood. The process of estimating the effort is therefore a continuous process, involving iteration and judgement. The following section will deal with the process of effort estimation and with the classic approaches used in this process.

Of particular interest is **'Who are the estimators?'** Almost certainly they are

not a homogeneous group occupying identical positions and carrying out identical tasks in organisations. Rather estimators come from all ranks; project managers, programme managers and software engineers all take part in the estimation process. Their needs are many and the task of building support for estimators is further complicated when considering **'when'** estimation takes place.

It is essential to know what total effort is involved in the development of a software project throughout its life cycle. The need certainly exists at the project initiation stage. However, this cannot be met because little is known about the product. All that can be done is to make an intelligent guess as to the size of the product, based on past experience. Boehm [Boe81] suggests that the estimates will deviate by a factor of 4. In other words within 80% confidence limits, the estimate will fall within a factor of 4 on either side of the final outcome.

Upon completion of the feasibility study an attempt can be made to supply better estimates, although the level of uncertainty will still be high. At that stage, the range of the estimates diminishes to a factor of 2 in either direction. The scope and objective of the system under consideration are known at that stage, as are the general design features. Issues such as the specific functions to be performed and how they should be performed, or even the specific types of user query to be supported are still to be pinned down.

The project manager acquires and accumulates knowledge which allows him to refine the early estimates by taking account of actual values. Estimates are further updated when the software requirement are specified, at the Preliminary System Design when preparing the work plan towards the Construction phase. At that time the user requirements as well as suggested ways to resolve them should be well known to the project team. Boehm suggests that the estimated costs could now fall within factor of 1.5. Figure 2.3 depicts the accuracy of software cost estimates for each phase. Rubin [Rub85a] reports a faster drop off in the level of uncertainty when using an interactive macro estimation procedure. That is, the estimation process begins by using a few key variables to forecast macro project characteristics and goes on to incorporate greater detail of a lower level to forecast micro project characteristics.

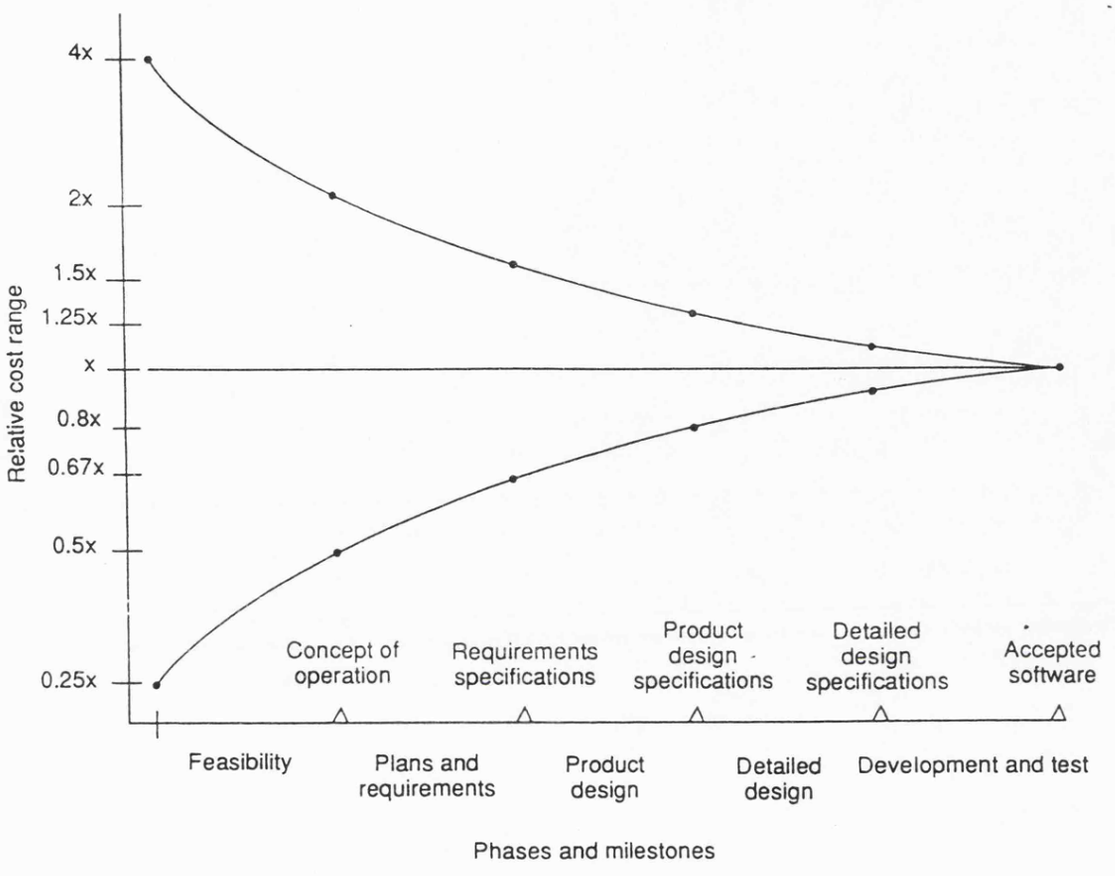


Figure 2.3 Software cost estimate accuracy versus phases [Boe81].

The nature of the task of estimating effort relies heavily on the judgement of experienced professionals, who know what is involved in the process of building software and are capable of applying their past experience into the estimation process. The process of effort estimation involves both analysis and synthesis. A project can be broken down into tasks that are analysed separately and then synthesised into an overall estimate. It is also an iterative process. However, at present, resource estimation is a creative art which is not applied in large scale software environments.

2.6 TOP-DOWN VERSUS BOTTOM-UP ESTIMATING

Essentially, there are two approaches for estimating of the quantitative product and project attributes: the top-down and the bottom-up. When the estimator approaches the process by applying his judgement to the overall system effort, by comparing it to the effort of similar past projects, the approach is referred to as top-down. The total development effort is divided up over the phases. In the case where the estimator chooses to base his judgement on a breakdown of the project into relatively small work units, to estimate them separately and then calculate the cost of the overall system, the approach is referred to as bottom-up. The unit breakdown is done to a degree that allows him to clarify the steps and the skills involved in completing the task and to identify similarities and differences between completed projects. Work units which are not comparable must be estimated by other methods.

Items of information used in adjustments for differences between project environments, include:

* Analysis of initial requirements.
* Type of software to be developed.
* User environment.
* Complexity and risk involved in the project.
* Programming technology used, languages and tools.
* Technical experience of development staff.
* Size of software product.
* Length of development.
* Number of development staff.

Thus, estimating software development can be implemented in either or both of these ways. The two approaches are complementary, however, the top-down approach is the only possibility before a detailed work breakdown structure of the project is available.

The advantage of a top-down approach springs from focusing on functions

at the system level. Thus, functions such as integration, documentation quality assurance or configuration management are considered, which might be neglected when implementing a bottom-up approach. The **pitfalls of a top-down** approach include the overlooking of technical difficulties and a lack of details needed for future cost justification.

The advantage of the bottom-up approach is looking into the system components in detail and can result in improved estimates. Furthermore, this approach enables us to allocate the estimation activity to the person who will be assigned to implement the task. This can result in a greatly improved commitment to the estimates. **The pitfalls of the bottom-up approach** stem from the accumulation of errors. Each piece of work being estimated includes some degree of uncertainty and inaccuracy. Therefore, the calculated overall effort will accumulate a high degree of error. However, this error is sometimes balanced out if the errors are distributed equally in conflicting directions and sizes.

2.7 ALTERNATIVE ESTIMATION APPROACHES

The alternative approaches for estimating the software effort and cost are:

* Expert judgement.
* Analogy.
* Parametric Models
* Standard estimating and Ratio Analysis.
* Parkinson's law
* Price-to-win

Apart from the last approach, any of these can be applied to a project as a whole (top-down), or as individual tasks (bottom-up).

2.7.1 Expert Judgement

One or more local or external experts are asked to estimate the required effort for a project. The experts rely on their experience with similar projects and use their intuition. This expert judgement can be implemented in various ways using different approaches and depends on the estimation objectives and the state of the project. Some researchers argue that the estimation process could be improved through using group consensus techniques such as Delphi [Boe81] or Decision Conferencing [Phi87]. Using more than one expert involves either averaging hence biasing the results, or grouping the experts and using group dynamics techniques to obtain an agreed estimate. The question arises in this context of how to average the individuals' estimates. Should the years of experience be considered as the important factor, which adds more value to the estimates suggested by an experienced programmer than to those suggested by a novice project leader? The literature does not support this view.

2.7.2 Analogy

It is frequently said that there is nothing new under the sun, which in this context implies that no new system is completely new. We can always learn from past experience, identify the resemblances or differences and make use of the known cost components, to support the required cost estimates. Using projects histories, similarities and differences as far as the effort estimation of the new project can be identified. Differences between the projects might be in either the development cycle or in system functionality. Estimating by analogy involves a form of pattern matching (reasoning) by analogy with completed projects or tasks. Wolverton suggested refining the first top-down estimates by reference to the more recent successful projects which include activity, duration and costs [Wol74;84].

Advantages. Estimating by analogy is a version of 'expert judgement'. Hence, 'expert judgement' assisted by analogy to similar representative projects might have advantages which stem from basing the estimate on recent experience

with projects or tasks successfully completed .

Disadvantages. There is a risk involved in this approach, which stems from the way it is used. Analogy is developed by intuition. Items which are not really related become associated out of the context of the considered issue. An additional weakness of the approach arises from the fact that it is often not clear which parts of the completed projects are represented in the new project. Although analogy is the approach most used for estimating the required effort for product development, it is the least definable in terms of mathematical or statistical rigor. Analogy requires an estimator to have not only a thorough understanding of the developed product, but also an in-depth knowledge of completed projects which are similar in functionality to the developed product. However, it is very difficult to retain the organisational knowledge of completed projects as a result of staff turnover and effort required to establish a thorough historical database. In addition, the development environments may differ between projects in the same organisation.

As the type of information needed by parametric models may not be known when software estimates are essential, and estimating by analogy may be the only viable alternative.

Artificial Intelligence, in particular the Expert Systems approach has recently been pressed into service for use in estimation by analogy. Estimation by Analogy using an Expert Systems approach has been proposed by Cowderoy and Jenkins [Cow86;88a], Galashan [Gal86] and Najberg [Naj88].[13]

All authors propose the incorporation of Expert Systems techniques for selection and evaluation of analogous systems or their components. The specification of the analogy selection criteria and technical definition of the project which requires an estimate are compared against the historical database

---

13. The system proposed by [Naj88] aims to support avionic systems, with an embedded software component. This system identifies the analogy by using one of the following approaches:

- Tolerances, which govern whether a certain data element value is equivalent.
- Weights associated with specific data elements which indicate their relative importance.
- Thresholds which determine whether an analogy exists or not.

for qualitative and quantitative attributes.

2.7.3 Parametric Models

The parametric modelling approach uses historical data to formulate one or more algorithms which produce estimates of software development effort. This approach makes use of an average 'productivity factor'. This assumes productivity as the basic factor, whereas software factors and organisational variables such as project team composition are somehow incorporated in the calculated productivity rate. Most researchers agree that there is a need to modify the results of the models for the 'software factors' which are identified as potential amplifiers of effort.[14]

The quality of the parametric models is highly dependent on an expert estimation of the size and complexity of the individual components of the system to be built. The underlying assumption is that the components of a model can be estimated more accurately than the effort needed for the development of a system. But, size is nearly impossible to predict with any degree of accuracy in the early stages of a product SDLC. Even as the product matures, this difficulty remains.

The historical data, from which the parametric relationships were extracted, affects the quality of the models. The parametric models are inherently linked to the conditions of their development environments. Any variance exhibited by effort estimation models is not so much due to a difference of perspective among the developers as it is to the unique nature of the software effort estimating task itself and more specifically to the dynamic environment that inherently governs the world of software development. It is the environment that is being modelled and not simply the required effort of a particular product.

Therefore, parametric models may be applied successfully only by analysts who are very familiar with the requirements of the product to be developed and with the characteristics of the development environment.

---

14. [Wol74;84]; [Her77]; [Wal77]; [Frei79]; [Boe81]; [Alb79;83]; [Jon83]; [Rub83]; [Jen84].

2.7.4 Standards Estimates and Ratio analysis

The 'standards estimates' approach is classified as a parametric model. This research is based on this version of parametric modelling. This approach to the estimation of software development effort is derived from the engineering world and in particular the standardisation approach to production planning and scheduling. Wolverton suggested a Standards Estimating approach, in which *"the estimator relies on standards of performance that have been systematically developed, and have become a stable reference point from which new tasks can be calibrated by ratio and or by similarities"* [Wol74;84].

It can be argued whether or not an analogy to the production line and hence to the production planning and scheduling procedure is appropriate for the software development process as a whole. This results in the view that a given project is very rarely repeated, which inhibits project to project comparison. Furthermore, the means of production is mainly human beings and not machines. Nevertheless, an insight might be gained from analysing the software development process, the various procedures, their components, and the ways they interact and are integrated, whilst bearing in mind concepts derived from engineering practice such as the 'Bill of Materials'.

The route in which the activities are performed in the SDLC is not rigidly defined. Various legitimate strategies are available for the development process, in which the activities take different forms as a result of changes in emphasis and therefore they are not always processed in a fixed sequence.

Ratio Analysis involves measurement of size and complexity at the module level. Both 'ratio analysis' and 'standards estimating' are adopted by the TRW model which deals with the estimation process mainly at the tendering stage. These methods have had some influence on the US DOD standards for software project management (DOD 2176 and 2176a).

2.7.5 Parkinson's law

G. Northcote Parkinson first described the phenomenon now known as *Parkinson's Law* or *"Work expands so as to fill the time available for its completion"* [Par57]. A manager may reduce the estimated effort and schedule to avoid the Parkinson syndrome and as a result the effort estimates and schedule might not be achievable. But, an alternative scenario might occur, such as the estimation experiment performed at the Rand Corporation, were 4 groups were asked to estimate identical software specification, using Delphi or 'standard group meeting'. The group which used the Parkinson approach, *("since there were 20 full time people available and they must finish in two years, or they don't finish at all")* estimated the effort to be 485 PM while the actual time spent on the project was 489 PM! [Boe81]. This approach cannot be considered as an estimating method. Macro and Buxton [Mac87] suggest that it is a *"a philosophy of despair"*. Yet, it is often the approach used in practice.

It is worth noting the results of the Jeffery and Lawrence [Jef85] survey which summarised the average productivity of the programming task by estimation methods. Jeffery and Lawrence did not observe that the estimation method had any impact on programming productivity as shown in table 2.1. Furthermore, they observed slightly higher productivity when the estimates were produced by a programmer alone (8.0) compared to the cases in which a supervisor estimated the effort without consulting a programmer (6.6). When an analyst estimated the effort, it correlated with even higher programming productivity (9.5) than the cases in which a programmer and/or a supervisor estimated the effort. These observations are quite reasonable. A system analyst is typically more familiar with the work in enough detail, and is more experienced than the programmer in effort estimation. However, Jeffery and Lawrence's last observation is quite surprising. The highest productivity was associated with no estimate (12.0 for 24 cases)! This result is not easily explained. They suggest that either these projects were simple and, therefore, estimates were not produced, or that tight deadlines prevented estimation and increased pressure. These results should cause us to question the Parkinson's legend.

EFFORT ESTIMATES PREPARED BY	AVERAGE PRODUCTIVITY	NUMBER OF CASES
Programmer alone	8.0	19
Supervisor alone	6.6	23
Programmer & supervisor	7.8	16
System analyst	9.5	21
No estimate prepared	12.0	24

Table 2.1 Productivity by Estimation Approach [Jef85]

2.7.6 Price-to-win

This estimation approach is followed sometimes when a company tenders for a project at a lower price than estimated, as result of an urgent need to win a tender, particularly when knowledge of the customer's budget figure has reached the tenderers. Managers who have the responsibility for software tenders believe that a price-to-win strategy is sometimes the only way to get a foot in the door of a company and by that, either cause a customer dependency on the software contractor, or afford him a better starting point for future tenders with this company. An inherent risk is often associated with this approach, resulting from inability of the tenderer to deliver the product within the agreed terms. A different incentive to take the 'price-to-win' approach in software estimates may be a desire to obtain software contracts which might enable to keep the professional people in the organisation throughout the 'seven bad years'. Or even an essential need to do so which resulted of the inability to make employees redundant. Such an approach might be suggested as a result from *a priori* knowledge of the customer's budget. This may or may not be an appropriate approach to manage software development, however, when taken, should always be accompanied by procedures to reduce functionality such as the design-to-cost approach discussed earlier.

The last two approaches are associated with the psychology and sociology of the tenderers.

2.8 IMPLICATIONS FOR EFFORT ESTIMATION

The first part of this chapter analysed and criticised the various processes for software development. An understanding of the main activities incorporated in these processes, and of the major concerns of each of the phases were gained. The opportunities and benefits associated with the new paradigms were identified. Thereafter, the traditional methods for software cost forecasting were described and analysed. Yet, what can be learnt from this analysis that is helpful for modelling the process of estimating the effort for software development? What can be suggested as the basis for modelling the process of estimating the effort for software development? The future trends of software development, the basic assumptions and the primary suggestions, upon which the Effort Estimation Model (EEM) will be built, are the subject of the following paragraphs.

2.8.1 Future Trends

> *We are not trying to tear down the lamppost, we just want to build one across the street where the search for one's wallet may be more fruitful* [Cur87].

The future of the system development process lies in the incorporation of new paradigms in a systematic approach to software development by exploring the advantages each one offers.[15] However, our ability to estimate the effort required to develop software products is heavily affected by the strategy chosen for the implementation. If the term 'easy' can at all be used in this context, then it is easier to estimate the effort required to develop software when the traditional process is used. The difficulty in estimating the development effort when the

---

15. The Spiral model [Boe88] for software development is an example of such integration.

implementation process follows the new paradigms stems, firstly, from the 'trial and error' approach which characterises the new paradigms. Secondly, the traditional model for software development have been experienced for almost two decades, and some understanding have been gained from observations of its behaviour in varying situations. As yet, there is not enough observations of the effort behaviour in the new paradigms, although it is observed that the effort required for the development process differs between of these strategies. The traditional model follows a pattern similar to the Rayleigh distribution [Nor63]; [Put78;79], the Prototyping and the Incremental development follow a pattern more similar to the Pareto distribution [Fox82[; [Ste87].

2.8.2 Base Model for effort estimation process

The traditional model for software development and the new paradigms evolve in the same principal phases as discussed early in this chapter: the Planning, the Preliminary System Design, the Construction, and the Operational phases. The basic elements are common to both frameworks, namely the activities and the intermediate products which are delivered at the end of a phase or a group of activities. Although these elements take somewhat different forms in each of the models, the similarities are of value for modelling the software building process and hence for the modelling of the effort estimation process.

This thesis suggests that the management framework for software building can be described as a base model. The manager of a software project can delete activities not relevant to the project under consideration, or add new activities to the base model. Changes to the activities and their components should be accompanied by a recorded explanation. In this way, knowledge associated with how projects are being implemented will remain in the organisation and can serve as the basis for comparison between the processes of project development, for the purposes of effort estimation. This knowledge could assist in training novice estimators in their first steps as estimators. They could learn what work breakdown structures are commonly used, and what the reasons are for changes.

The base model should represent activities of a range of particular software processes and should allow reasoning about their use. It should include the drivers for the effort required for each activity, and the standard level of effort needed to implement each of the cost drivers and their associated activities. Recently, much attention has been given to modelling the process of software development, with emphasis not only on the products of this process, but on an approach that will make it possible to reason about the software process and provide the basis for structuring automated software environments [Cur87]; [Leh87;87a]; [Ost87]. However, satisfactory software process models have yet to emerge [Wil88].

2.8.3 Phase-based estimation

The Preliminary System Design phase is critical to all the parties interested in the process of the software development, in particular to management and users. Sufficient information resulting from problem analysis and definition of the system is accumulated throughout this phase and thus provides a reliable basis for estimating the costs and benefits of proceeding with the development. However, the time required for this phase is often underestimated and does not include the effort consumed in studying the application domain.

Defining system specifications is as much a political as an intellectual process. Thus, involving users in the design process by using the new paradigms as catalyst for communication encourages creative participation and stimulation of suggestions. This process may eliminate the emergence of conflicting goals. It may help keep users' expectations in line with realistic opportunities, which is a very important target of system development and one of its prime critical success factors.

The belief of the author of this thesis is that estimating the effort needed for project development at the Project Planning phase, can be done relatively accurately **only for** the subsequent phase in the project life cycle, the Preliminary System Design. However, from these estimates coarse estimates of the total development effort can be extrapolated. This should be based on statistical data

with regard to the resource distribution between the PSD and the rest of phases of the software development life cycle.

Only towards the end of the Preliminary System Design phase, better estimates for the Construction phase and for the total development can be offered. This can be done as result of the accumulated knowledge about the requirements, the design possibilities and the actual performance, which allows us to validate our estimation assumptions and to correct them (as discussed in Paragraph 1.6.1) This should be an ongoing process throughout the development process, as summarised in Figure 2.4.

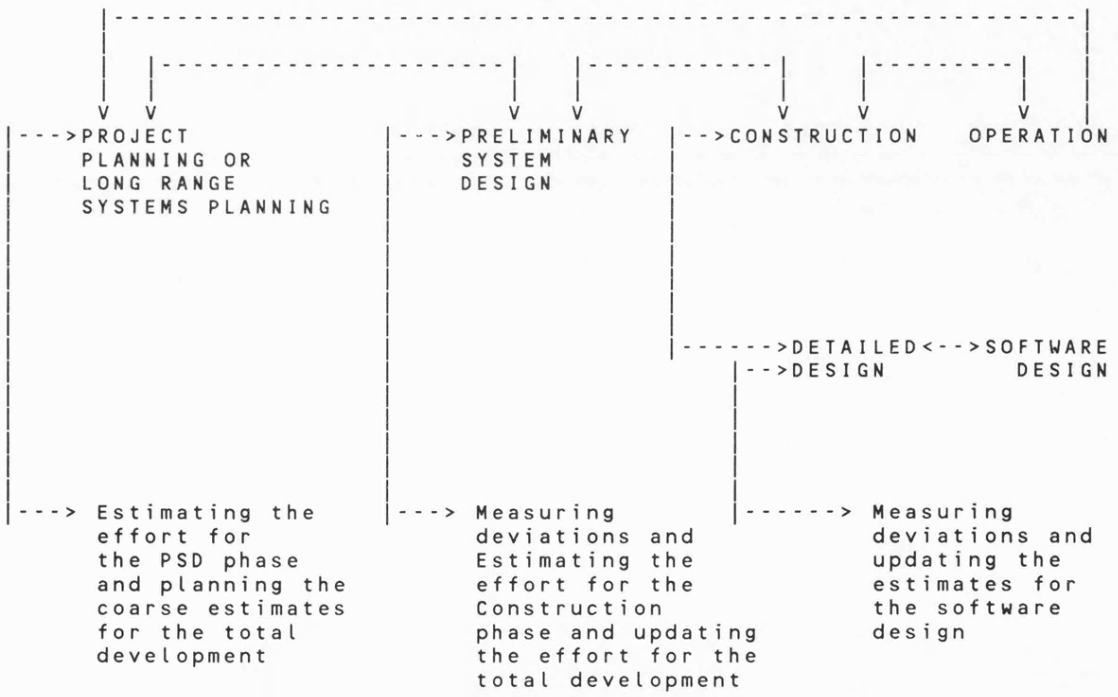


Figure 2.4 The interaction between software development and phase-based effort estimation processes.

The discussion in this chapter leads to the conclusion that the PSD phase is of great importance for the development process. Using the new paradigms does not reduce its importance, although using them influences our ability to cope with

some of the more problematic areas associated with this phase. The skills required to develop this phase and the triggers of its costs are different from those required for the Construction phase. Therefore, the proposed EEM is a phase-based estimation process.

This thesis argues that on the basis of the knowledge which exists in the organisation early in software development, it is feasible to offer a process which will assists us in:

- Evaluating the complexity and the risk associated with the development process as a whole, and thus enable us to plan the resources needed for it.
- Estimating the effort associated with the Preliminary System Design phase only.
- Planning coarse estimates for the effort needed for the total development.

The hypothesis in doing so is that the EEM will yield estimates with a high degree of accuracy, for the effort required for the Preliminary System Design phase. However, even though the degree of precision of the coarse estimates for effort needed for the total development will be less accurate then those estimated for the PSD, it is presumed that the procedures by which the EEM will lead the project manager through the estimation process will be beneficial to him. Such a process will help the user to understand the meaning of the suggested estimates, the assumptions and the measurements they are based on. The effort needed for the rest of the software development phases should be estimated based on different cost drivers and may use units of productivity measurement different from those suggested for the PSD estimation.

2.8.4 Judgement and Measurement: on the horns of a dilemma

The two fundamental methods followed in estimating are expert judgement using analogy and parametric models. As discussed early, the expert judgement approach may be incorporated in parametric models. Most of the models, if not

all, require expert judgement throughout the estimation process. Expert judgement is essential in the process of sizing the product to be developed. It is of particular importance in projects which are characterised by high degree of innovation and, therefore, high degree of uncertainty and, thereby, complexity. Judgement is essential in evaluating the effort required for sub-processes which cannot be compared with any projects which have previously developed in a similar environment. Analogy is used to identify similarities or differences between components of products or between complete products. Some models facilitate an iterative process to obtain the estimates and some have an associated tool which supports the product size estimation process.

Cost estimation, by its nature, is not an exact science. Only a certain level of accuracy and precision is possible. Many factors affect the software development process, differing between organisations, and varying in the degree to which they affect the development process. The estimation process involves assumptions and judgements and, therefore, carries inherent risk, particularly early in the project, and *"all efforts to do better are futile"* [Put79].

Indeed, the spectrum of opinions on that matter is wide. Some philosophers, researchers and practitioners take the attitude of the scientist as expressed by Lord Kelvin:

> *When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind* [Lord Kelvin, 1889]

Others will argue that if something cannot be measured in an exact and precise way, it is not worth measuring at all. They will argue that a random number is as good as the imprecise measurement, as:

> *all our reasonings concerning causes and effects are derived from nothing but custom. .... Or, there is nothing in any object, consider'd in itself, which can afford us a reason for drawing a conclusion beyond it; and, That even after the observation of frequent or constant conjunction of objects, we have no reason to draw any inference concerning any object beyond those of which we have had experience* [David Hume, 1739]

Engineering disciplines are marked by the availability of standardised, precise and well understood parameters of measurement that have evolved through a process of scientific examination and investigation, such as the measurements of length, pressure and temperature. These measurement are based on physical properties. But our domain of interest, the software science, does not have such metrics. There are good reasons for this phenomenon. There is no agreed unit of measurement for a software product, from which productivity can be measured and used for the effort estimation process. In its conventional economic definition, the term productivity is quoted as the ratio of the amount of product over a given input of means of production, e.g. labour or expenses. This definition implies the need to measure the amount of input and output of the production process. Measuring 'something' requires comparison with a standard. But, what could serve as the standard unit of measurement for a product in the software arena? What units of measure should be used to measure the amount of input and output involved in the production process? The issue is easier for the input part. In measuring the manpower effort, the input measure will be person-hours (months/years), even though the people involved in the process do not have identical characteristics and, therefore, could not produce the same quantity or quality of a defined product unit. However, the issue is much more complex when the unit of measure for the software product is considered. This issue will be further discussed in Chapter 4, after investigating the current practice.

It is crucial that a single set of underlying principles generally serves as the basis for many systems, in a broad variety of contexts. This modelling paradigm applies to the soft sciences as well as to the hard sciences. The difference is in essence that in the soft sciences the problem are generally of statistical or probabilistic nature. The cost is high for collecting and analysing data of project performance, at a detailed level and precision which allows systematic comparison. Most of the efforts of data collection is esoteric. The results of the analysed data are given in literature. However, the detailed databases are, in most cases, proprietary and are not available for further examination. As a result, comparison between software development at the abstract level is rare. Software engineering does not yet have, adequate rules of standards practice.

The thrust of this research is that both measurement and judgement, should be used in modelling the effort estimation process. The estimator should:

- Based his estimates on measurement of what can be measured.
- Use his judgement to assume values for attributes that either cannot be compared to other projects, or have no hard data associated with their performance.
- Use his judgement to assume values for attributes which the available data is associated with high degree of uncertainty. This uncertainty might, hopefully, be reduced when the process is repeated through the software development process.

Therefore, Aristotles' philosophy should be adapted:

It is the mark of an unstructured mind to rest satisfied with the degree of precision which the nature of the subject admits and not to seek exactness when only approximation of the truth is possible.

Attributed to Aristotle, 330 BC.

This thesis' suggestions for modelling the estimating process are based on the principles discussed in this chapter. However, before we allow ourselves to elaborate on the subject, we have to investigate the models and tools currently in practice for effort estimation and system sizing purposes. These are the major themes of the second part of this thesis.

PART II - STATE-OF-THE-ART

This part of the survey is comprised of three chapters which as a whole provide a thorough analysis and understanding of the current practice of modelling the process of estimating the effort required for software development. But, models for estimating the software effort assume that project size can be estimated. Therefore, these two major themes are the subject of the first two chapters. Chapter 3 focuses on models for estimating the effort required for software development and Chapter 4 on models for estimating the size of software product.

Effort and cost models have been developed since the late 1960's, but, models and tools for sizing the software product have been developed only over the last few years. The need for the size models was always there. However, this latest development is attributed to the advances made in the fields of Formal Methods and Artificial Intelligence, mainly Expert Systems. A representative of this group of models will be analysed. The implications of errors in size estimating on predicting the effort required for development will be addressed. A point in question is what could be considered as an appropriate unit of measurement for estimating the product size, in particular, for the purpose of estimation the effort required for the PSD phase, which is the scope of this research.

Estimating the effort for software development cannot be isolated from research in adjacent fields of interest. Indeed, in order to understand the observed behaviour of the software development effort, researchers investigating the Software Development Life Cycle (SDLC) have approached the subject from different directions, as discussed in Chapter 2. From this discussion and the analysis of the themes addressed in Chapters 3 and 4, several foci of interest for effort estimation have emerged. These are the aspects of uncertainty, complexity and the resource distribution among the phases of SDLC. In Chapter 5 these issues will be addressed, as well as additional points of interest to support the EEM.

Only a thorough discussion will enable us to support firmly the Effort Estimation Model (EEM) proposed for early stages of software development.

Chapter 3

SOFTWARE EFFORT AND COST ESTIMATION MODELS

3.1 INTRODUCTION

Chapter 3 introduces the ontology of effort estimation models for software development and their evolutionary development. Emphasis will be placed on models which are regarded as 'classics' in the area of modelling the estimation process of software development. The models presented in this survey contributed either in establishing the foundation of a new theory or improving the practice. Their underlying assumptions, the methodology used to derive them, and their strengths and weakness will be analysed. Discussion and critique will follow the description of each model. The chapter culminates with a comparison of models with a particular focus on the comparison between the representatives of the two main approaches currently in practice. They are the SLIM [Put78] tool and the COCOMO [Boe81] set of models. SLIM is based on Norden's [Nor63] and Putnam's analytical model [Put78], whilst the COCOMO represents the composite modelling approach for effort estimation.

The analysis focuses on the following themes which are of particular interest to this research, namely:

* Methodology in which the models were achieved.

* Estimation approach, e.g. top-down or bottom-up.
* Factors affecting productivity, i.e. complexity determinants.
* Resource allocation among the phases encompassing the SDLC.
* Results of applying the models.
* Applicability of the model to the various SDLC phases.
* Estimation approaches to product size and what should be considered as an acceptable standard measurement for unit of product.

3.1.1 Estimation models and tools; What do they provide?

The existing effort and cost models vary in the range of facilities they offer. Based on product size, most provide estimates for manpower resources, effort and duration. Some models incorporate a 'what-if' analysis. Some add an effort - to - cost conversion which may include inflation and financial cash flow factors. Some have an attached planning tool which enables the project manager to select the preferred strategy for software development, including the associated activities for the development effort. Some have an attached or a stand-alone model supporting the sizing process.

The tools differ not only in what they offer and in the methods employed to obtain them, but also in their data collection, type of projects and the environmental factors among software development sites included in the database upon which the model is based.

3.1.2 Classification of Effort and Cost Estimation Models

Several researchers have suggested classifications of resource and cost estimating models. Shooman [Sho79] differentiated between resource models and cost estimating models. A similar approach was taken by Jeffery and Vessey [Jef80] when they compared Putnam's model to the one developed by Walston and Felix

[Wal77]. Kitchenham and Neumann [Kit89] distinguished between models that specify the relationships among various cost parameters (e.g. effort/cost and or duration/staff level) and models that do not. It is in the interest of this research to understand the methods used in deriving the models. It is, therefore, appropriate to adopt the Conte et al. [Con86] categorisation of effort and cost models, depending on the method used in deriving the models, namely:

* Statistically based models.
* Historical experimental models.
* Theoretically based models.
* Composite models.

3.2 STATISTICALLY BASED MODELS

Linear models have the form:

(3.1) $E = c_0 + (SUM\ c_i x_i)$

Where,

E = effort in person-months.

c_0 = fixed cost per system, usually derived from historical data.

x_i = cost driver, i.e. software attributes that are believed to affect software development effort.

c_i = cost associated with the specific cost driver. The values c_i are chosen to provide the best fit to a set of observed projects.

Non-Linear models are expressed in the form:

Either,

(3.2) $E_{(nom)} = a + b\ S^c$

Where,

$E_{(nom)}$	=	is the nominal effort before applying an adjustment multiplier.
S	=	estimated size of the project in KLOC. (Thousand lines of code).
a,b,c	=	constants usually derived by regression analysis of historical data.

Or,

(3.3) $E = (a + b\,S^c) * M(X)$

Where,

$M(X)$	=	adjustment multiplier that depends on one or more cost drivers, given in a vector X.

3.2.1 The System Development Corporation (SDC) Model

This pioneering model was developed in the mid-1960's by the System Development Corporation (SDC) group and described by Nelson as early as 1966 [Nel66]. The study included 169 projects. Of the 104 cost drivers, only 14 were classified as leading. The sample included a variety of applications, large and small projects, written in assembly and high-level languages. The study analysed programming effort only. Design, integration and testing were excluded, as was management effort.

The model equation takes the specific form of *(3.1)*:

$$E = -33.6 + 9.15x_1 + 10.73x_2 + 0.51x_3 + x0.46_4 + x0.4_5 + 7.28x_6 + (-21.45x_7) + 13.53x_8 + 12.35x_9 + 58.82x_{10} + 30.61x_{11} + (-0.53x_{13}) + (-25.20x_{14});$$

And the attributes found to affect the software effort are:

x_1	Lack of requirement	
x_2	Stability of design	(0-2) *
x_3	Percentage of mathematics instructions.	(0-3)
x_4	Percentage of storage and retrieval instructions.	
x_5	Number of sub-programs.	
x_6	Programming language.	(0-1)
x_7	Business application.	(0-1)
x_8	Stand alone programs.	(0-1)
x_9	First program on computer.	(0-1)
x_{10}	Concurrent hardware development.	(0-1)
x_{11}	Random access device used.	(0-1)
x_{12}	Different host target hardware.	(0-1)
x_{13}	Number of personnel trips.	
x_{14}	Developed by military.	(0-1)

* The numbers in the parentheses refer to rating to be made by the estimator.

The major cost drivers affecting the software effort are: lack of requirements, stability of design, concurrent hardware development, random access devices used and changes in host target hardware.

Discussion

Assigning zero values is a legitimate option, and the model results in an effort of -33.6 PM to produce nothing [Kit89]. Linear models have been found to correlate quite satisfactorily with development effort for small to medium size projects [Jef79]. Yet, Boehm [Boe81] and Conte [Con86] concluded that linear models were unsatisfactory for effort estimation. It is common to assume an exponential increase in effort with size of project for very large projects. A possible interpretation being that the effort is a highly non-linear function of a large number of variables.

Project size measured by lines of code (LOC) does not enter into the model as a direct factor affecting the effort, even though all other models assume effort increases with program size.

Boehm [Boe81], Conte [Con86], Macro and Buxton [Mac87] all agree that the SDC model does not perform well by evaluation criteria such as mean estimate and standard deviation.

The SDC model is mainly of historical interest. The statistical methods by which the model has been obtained (least squares and factor analysis), the significance of the parameters indicated, and the data associated with programming productivity rate provide a valuable basis for further research into parametric effort estimates and productivity, assuming linearity. One such model is by Walston and Felix [Wal77] which will be discussed in Paragraph 3.3.2.[1]

3.2.2 Aron's Model

Aron [Aro69] of IBM introduced a new concept to the software development taxonomy: complexity. He interpreted complexity as a function of the difficulty of the project and its duration. He suggested that difficulty is mainly caused by the interactions of the project under development with other projects. Aron also determined a matrix of productivity rates that explicitly recognised this, and provided guidelines on classifying difficulty.

- **Category 1:** An 'easy' classification is characterised by very few interactions with other systems. This difficulty level applies to programs that generally interact only with input/output programs, data management programs and

---

1. A similar approach was taken by Farr and Zagorsky [Far65]. Some of the attributes they believed to be of importance were different:
   * Number of instructions.
   * Number of document types delivered.
   * Experience of systems programmers.
   * Number of display consoles.
   * Percentage of new instructions.

monitor programs.

- **Category 2:** Is classified by the existence of some interactions. The programs included in this category are mostly utility programs, compilers, input/output packages, schedulers in management packages.
- **Category 3:** The third category is characterised by many interactions with other system elements. Monitors and operating systems fall into this category.

Aron suggested the productivity table (see Table 3.1) to assist the classification process. He recognised that the manpower employed in producing large systems builds up gradually, reaching a peak close to the time in the project development life cycle when the system test is completed. This phenomenon was later observed by Putnam also. Aron suggested the following four criteria for identifiers of a large system:

* Size of project team, if more than 25 programmers are involved in the development process.
* Development time is more than six months.
* Levels of management, if more than one level of management is associated with the process.
* Program size is more than 30,000 deliverable instructions.

	Duration		
Difficulty	6-12 MM	12-24 MM	more than 24 MM
	Instructions per		
	day	month	year
Easy Very few interactions	20	500	10,000
Medium Some interactions	10	250	5,000
Difficult Many interactions	5	125	1,500

Table 3.1 Aron's Matrix of Productivity Rates

Discussion

It should be noted that these identifiers are still valid, although the values for some of the attributes may differ today. Interfaces between sub-systems and anticipated duration of development are considered as a major cause of complexity and for the risk associated with software development. This type of complexity results mainly from uncertainty and entropy.

Aron's observations and concepts introduced in this work established the foundation for a software estimation practice. These are the:

- Interaction between project under development and other projects is the cause for project difficulty.
- Difficulty and duration is the cause of complexity.
- Standard productivity matrix and the identifier of large system[2]

3.2.3 Bailey and Basili's Meta-Model.

Another approach to software cost modelling was suggested by Bailey and Basili [Bai81]. They suggest a meta-model for effort estimation. The meta-model is a defined set of statistical procedures for creating a model from a set of software development project data for a given environment. Statistical analysis is used to produce a baseline effort estimate using final product size as input. Bailey and Basili discuss in detail effective methods of defining and statistically analysing the effect of various additional project attributes as correction factors to be applied to the initial baselines estimate. Bailey and Basili's [Bas81] equations for calculating effort are:

---

2. The concept 'largeness' is addressed by Belady and Lehman [Bel79b] when characterising large software systems. They argue that the root cause of **largeness** is the **variety** and not the number of instructions or the number of modules in a software product, although they are expected to grow as result of system largeness. They account for the variety of needs and activities associated with the development and maintenance of software. More on this subject matter in Chapter 5.7.4

(3.4) $E = a + b\,S^c$

Where,

E	=	total effort in person-months.
S	=	estimated size of the project in KLOC.
a	=	constant, the initial preparation to understand the design before programming begins.
b,c	=	constants

And,

(3.5) $E = 3.4 + 0.72\ DL^{1.17}$

Where,

DL	=	composite measure of work based on LOC. Different classes of programs were given different weights before they were summed.

Basili [Bas83] suggested that the constants a and b are not transportable to a different environment. Hence each environment needs to develop its own model for prediction of effort for software development. The importance of this model is in providing a methodology by which an individual organisation may build its own estimation model which is calibrated to its particular environment.

3.3 HISTORICALLY BASED MODELS

3.3.1 The TRW Cost Estimation Model

Of the historically based models that have been described in the literature, the TRW model is probably the best known. This is one of the few models that actually estimates the cost of a project in monetary terms. The model is classified as a static, multi-variable model [Bas83]. Wolverton [Wol74], recognised different effects on cost as a result of difficulty arising from the use of new or old code.

Assumptions

- The TRW cost estimation algorithm is based on the assumption that cost varies proportionally with the size of the software product, measured by the number of instructions.
- Various categories of software routines have their own cost of development. The re-use of software and difficulty are the dominating factors for the estimating process at the early design stage. Cost per object instruction is the only parameter that changes as a function of the degree of novelty and difficulty of the estimated project.
- Resource allocation among phases of the SDLC is based on historical data.

The estimation process

Producing the estimated cost involves the following steps:

1. Group the software routines into six categories:

    * Control routines. (Control execution flow)
    * Input/output routines.
    * Pre- or post-algorithm processor, (manipulate data for subsequent processing or output).
    * Algorithms which perform logical or mathematical operations.
    * Data management routines which manage data transfer within the computer.
    * Time critical processors, which are highly optimised machine dependent code.

2. Estimate the size and complexity by routine. Six levels of difficulty are suggested for each routine, based on the combination of degree of the re-usable software and difficulty factors, namely:

	Easy	Medium	Hard
Old	OE (1)*	OM (2)	OH (3)
New	NE (4)	NM (5)	NH(6)

* The numbers in the parentheses (1...6) indicates a form of order for the degree of difficulty involved in developing the various category of modules.

Then,

(3.6) $C(M) = S(M) * C_{i(M), j(M)}$

Where,

$C(M)$ = represents the cost of a module *M*.

$S(M)$ = the estimates line of code for a module *M*.

$i(M)$ = the module type (category).

$j(M)$ = the presumes difficulty of the module.

$C_{i(M), j(M)}$ = the corresponding cost by category and degree of difficulty.

The cost is the total number of instructions by category and degree of difficulty multiplied by the corresponding cost. The overall cost of the system will then be obtained by summing the cost over all modules:

(3.7) $\underset{\text{all modules}}{SUM\ C(M)} = System\ cost.$

3. Identify the development strategy, namely the various phases for producing the software from the conceptual stage to the delivery of operational software to the user.
4. Define the activities in each development phase by means of an activity array and the associated cost matrix.
5. Provide schedule data based the on user's need or other management considerations. The schedule data are input as months from go ahead, for each of the milestones in the SDLC.

Resource allocation among phases of development

Wolverton provides us with statistics of resource allocation among the development phases from few resources. He concludes that statistics are necessary for the assignment of the resource to each of the activities for each of the phases of development, however, judgement gained from previous experience is fundamental. The resource allocation distribution is used as a baseline which is then intuitively adjusted to predict the effort required for each of the phases. The resource distribution serves also as guideline to the staffing activities. The subject is further discussed in Chapter 5, including comparisons with data provided by other research.

The resource distributions are based on empirical data which varies greatly among particular projects, depending mainly on the complexity level of the specific project.

Discussion

As in Aron's [Aro69] work, this approach introduces a kind of objective measure of complexity, old and re-used code, although, the process for arriving at cost involves a great deal of subjectivity. The range of cost given varies from a low of $15 per instruction for easy algorithms and old code to $75 per instruction for all kinds of time critical processors.[3] Even though the values are given in monetary terms, the cost matrix is valuable and may be used as a complexity weighting for estimating module size. However, it should be noted that the matrix is applicable for use only after the project has been broken down into module level, i.e. quite late in the design process.

The database used for developing the software cost includes the sensitivity coefficients or exchange ratios which were used by the cost estimation algorithm.

---

3. The monetary values are corresponding the values of the US dollars in the late 1960's and early 1970's.

They reside in the database and are not available for further investigation.

3.3.2 Walston and Felix's Model

The work of Walston and Felix from IBM [Wal77] covers 60 projects, completed during the period 1973-1977, from which they obtained coefficients for their static single-variable model, using multi-linear regression analysis. They proposed a **Productivity Index** which takes the form of a static multi-variable model [Bas83]. The model was obtained empirically. The following data was collected from completed projects, by phase of software development:

* Number of modules.
* Number of pages of documentation.
* Errors.
* Use of modern programming techniques.
* Computer resources.
* Languages used.

Assumptions

The rate of productivity is a factor of the delivered LOC produced by a program. The factors affecting this productivity rate are measured to produce a Productivity Index, which predicts productivity for a new project and thus its effort.

The Model equations

The baseline effort equation is non-linear of the general form described earlier. Walston and Felix's basic equation for effort is:

(3.8) $E = 5.2\ L^{0.91}$

They suggested additional equations for the following relationships:

(3.9) $S = 0.54\ E^{6}$

(3.10) $DOC = 49\ L^{1.01}$

And either,

(3.11) $D = 4.1\ L^{0.36}$

Or,

(3.12) $D = 2.47\ E^{0.35}$

Where,

E	=	total effort in person-months.
L	=	thousands lines of delivered code (KLOC) including comments.
S	=	average staff size, i.e. total staff months/duration of the project.
DOC	=	pages of documentation.
D	=	Duration of the project in calendar months.

Factors affecting productivity

However, Walston and Felix found that their basic effort equation *(3.8)* performed relatively badly on their data sets, so they investigated the effect of a number of factors which might have been responsible for the large deviations between actual and estimated effort. Project managers were asked to rate the 29 productivity variables for the expected condition of specific projects. The average productivity was calculated for all projects with the same rating for a particular factor. The changes in productivity which could be attributed to this factor were calculated from this figure.

Walston and Felix identified user involvement as a key issue affecting

productivity and consequently software development effort. They concluded that experienced users could improve productivity whereas inexperienced users could affect productivity negatively. The productivity rate for the 'Customer Interface Complexity' variable, observed in this research, ranges from 500 DSI/PM to 124 DSI/PM (DSI, Delivered Source Instruction). The drop is due to time and effort lost from the need to communicate and co-ordinate with the customer. But, Fox argues that Walston and Felix could not conclude this from their database which is *"too sparse to draw any such conclusion"* [Fox82].

Three out of the eight attributes that most affect the productivity are related to the user interface, four to personnel skills and the eighth is associated with pages of documentation produced. Walston and Felix used the set of 29 attributes to define a Productivity Index.

The proposed Productivity Index

(3.13) $I = SUM\ W_i X_i$

Where,

I = productivity index. This can be negative or positive.

X_i = -1, 0 or +1, depending on the task rates (low, medium and high) with respect to certain attributes of the productivity variance factor.

1, if the rating for variable $_i$ increased productivity.
0, if the rating for variable $_i$ was nominal.
-1, if the rating for variable $_i$ lowered productivity.

W_i = Weighting defined as:

(3.14) $W_i = 0.5 \log_{10} (PV_i)$

Where,

PV_i = productivity range measured by the ratio between the lowest and the highest values found for variables $_i$.

By linear regression and least square they find the best fit to the equation,

(3.15) $Log\, L = a + b\, I$

Where,

L = project productivity.

Which can be used to predict productivity for a new project. The effort is obtained by:

(3.16) $E = S / L$

Where,

S = Lines of code

Discussion

The range of projects was diverse in terms of the range of LOC, (from 4 KLOC to 467 KLOC) with project durations of 12 to 11758 person months, languages (28 high level languages) and computers (66 computers) used.

The Walston and Felix model, as with the SDC model, does not use the actual size of the product directly. The size is used as a scaling factor to convert an estimated effort per line of productivity into an overall effort estimate. However, Walston and Felix did not suggest any transform ratio for this conversion. This is analogous to the TRW model which uses size as a scaling factor to convert estimated cost per instruction into an overall cost estimate.

The method has theoretical limitations and, as neither the database nor the actual model are given, practical limitations. The method assigns the values to the X_i's, as in practice there are environment and product related factors which affect them. This is considered to be a problem. The important features of the model

are its contribution to the list of factors affecting productivity and the method suggested for assessing the impact of each of the factors, on the programming productivity. It is also the case that IBM continued to refine their approach to productivity and effort estimation by continuing the collection of project data, and are now claiming extremely high precision for their cost estimates [Dal85].

The Walston and Felix model was the only exponential model which has exponent less then 1. The implication of the exponential value is that larger projects achieve greater productivity than smaller projects. In other words a model that exhibits this characteristic suggests 'economies of scale' while a model with an exponent greater than one suggests 'diseconomies of scale' [Boe81]. Jeffery and Lawrence [Jef79;81] and Jeffery and Vessey [Jef80] have reported 'economies of scale' on small projects.[4]

But, if one looks closely, it is almost linear and probably possible to establish a linear relationship, [Bro83].

(3.8) $E = 5.2\ L^{0.91}$

Indeed, Jeffery and Lawrence convert equation *(3.8)* to a linear one as follows:

(3.17) $Effort = a\ Size + b$

Where,

a and b are constants derived from historical data.

Freburger and Basili [Fre79] analysed a sample of 19 projects from the Software Engineering Laboratory.[5] They began with thirty factors and ended with only two major factors upon which effort is dependent:

---

4. This subject will be addressed in a more detailed form when introducing Boehm's set of models.

5. This work is based on the same data-base used by Bailey and Basili [Bai81]

* The organisation environment.
* The project team composition.

They reported the following relationships as applying to the Software Engineering Laboratory database:

(3.8a) $E = 1.41\ L^{0.93}$

(3.9a) $S = 0.24\ E^{0.74}$

(3.10a) $DOC = 30.4\ L^{0.92}$

and either,

(3.11a) $D = 4.6\ L^{0.26}$

Or

(3.12a) $D = 4.4\ E^{0.26}$

Basili [Bas83] further argued that the relationships arrived at by Walston and Felix are not necessarily transportable to another organisation. DeMarco labelled this *"Waiting for GODOT"* [DeM82]. He concluded that each environment should develop its own model for software development effort prediction and, indeed, suggested a method so to do.

DeMarco suggested that the results of this model are acceptable because the convergence was obtained even before adjustment. Boehm suggested that this form of model appears to work well if the variables chosen are reasonably independent. Otherwise, a problem of double counting of the cost and interaction effect arises.[Boe81].

3.3.3 Doty's Model

Another model which falls into this category is the Doty's Model. Doty [Dot77] conducted a cost study for the US Air Force in which the program domain is

divided into four categories based on the language used. He hypothesised that productivity increases with high level language due to reduction in size. He found that the productivity increased by a factor of three to seven. The Doty factors for productivity modifications are:

* Special display (1.11-1.43);
* Detailed users requirements (1.11-2.0);
* Volatility of users requirements (1.05-1.05);
* Real time (133-167);
* CPU constraints (1.18-1.43);
* Time constraints (1.33-2.32);
* New hardware (1.92-1.92);
* Parallel hardware development(1.25-2.22);
* Remote devices (1.43-1.43);
* Site devices (1.39-1.39);
* Host Devices(1.11-2.22);
* Multi-site device (1..21-1.75);
* New languages (1.80-1.80);
* Interactive devices (.83-.83);
* Software engineering access (.67-.90).

Doty provides a different set of weights for different applications. This study covered estimated effort required for analysis and development but not maintenance and enhancements.

3.4 ANALYTICAL MODELS

3.4.1 Background - Norden's Model

One of the most influential milestones is the work of Peter Norden [Nor60;63]. This is recognised as the foundation of the analytical approach based on the plausible mathematical relationship between project team size and phase of activity in project development. A few of the major contributors to the research in this area followed Norden's findings. Norden studied sixty hardware research and development projects at IBM and concluded that:

- Projects are composed of overlapping work cycles or phases.
- There are regular patterns of manpower build-up and recurrent patterns for these phases, which can be fitted into a Rayleigh distribution. There is a family of such curves, relating manpower used each month with elapsed time. The curves are fitted to a small number of successive 'cycles' of work which occur during the life of a project. The cycles differ only in size and proportions as shown in Figure 3.1. Each cycle can be described by the following equation:

$$Y' = 2Kat\,[\exp(-at^2)] \qquad (3.18)$$

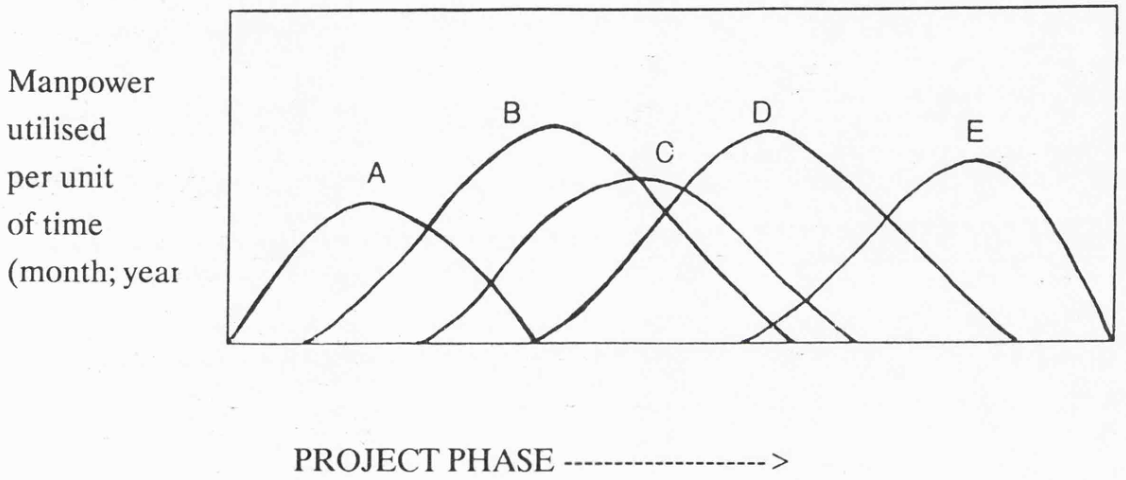
Where:

Y'	=	manpower utilised each time period measured in PM.
K	=	total cumulative manpower utilised by the end of the project.
a	=	shape parameter (governing time to peak manpower).
t	=	elapsed time from the start of cycle measured in PM.

When the individual cycles are linked together, they produce the profile of the entire project. *"The life cycle manpower model provides an orderly summary,*

crystallised out of past experience, at functional level. It gives us the capability of conveniently using historical experience to shed light on the future" [Nor63].

The manpower cycle curve is computed either by using the foregoing relationships or by statistical curve-fitting techniques.[6]



A.	Planning Cycle	D.	Design Cycle (extensions)
B.	Design Cycle	E.	Product Support Cycle
C.	Model Cycle (prototype)		(Modification and maintenance)

Figure 3.1 R&D projects are composed of cycles

The staffing of engineering R&D projects is not inherently a matter of wide tradeoffs of time and manpower. Rather, they seem to embody a two-stage process in which the limiting condition is the problem-identification rate or insight environment of unsolved problem space. This implies that one cannot indefinitely add people and get the job done faster [Nor 63].

---

6. The fact that this distribution has a 'tail' explains the ninety percent work completion syndrome.

This conclusion was confirmed later by Brooks, who claimed that *"putting more people on a late job makes it later"* [Bro75]. Norden found that the limiting factor is the rate in which ideas can be generated, and that this is not widely affected, **if at all**, by the number of staff, but rather by some capability level of the group. Generating ideas and solving problems are random and independent events. Therefore, within one cycle (phase) the exponential parameter remains constant over a considerable period of time.

The relationships among the Rayleigh parameters are highly complex. This probably explains why purely empirical approaches have not yielded satisfactory solutions. This issue will be elaborated while addressing the SLIM tool for effort estimation, which is based on Norden's findings.

3.4.2 Putnam's Model, SLIM

Putnam's Resource Allocation Model (SLIM) is based on the work of Norden [Nor63], Aron [Aro69], Brooks [Bro75] and Putnam's own findings.[7] He analysed several hundred of software projects in the Computer System and Command and other US governmental organisations. The model specifies the relationships among effort, duration and staffing and describes the variation of staffing level during the project life cycle. The model is classified as dynamic, analytical and multi-variable since the Rayleigh curve describes the variation in manning level across the development time.

Methodology

Putnam's analysis uses statistical and mathematical approaches that incorporate linear programming for determining optimal values of objective functions in the light of known constraints. One of the major relationships employed by the SLIM model, the productivity - difficulty relationship, was found empirically by plotting

---

7. [Put75;78;79;80;80a;81;84a]

difficulty versus productivity lines from which the Software Equation was manipulated.[8]

Assumptions

The basic assumptions of the SLIM model are:

- The manpower utilisation of software development follows a non-linear form which matches the Rayleigh curve and Norden's theoretical approach to problem-solving behaviour. Complexity factors which are incorporated are based on interactions of the project team as they carry out the connected activities which formulated the project.
- The shape of the curve somehow relates to both the difficulty of a particular development (state of technology incorporated in the project work) and the skills of the project team.
- The Software Life Cycle is dynamic and not static.
- The number of problems to be solved is finite.

It should be noted that Putnam explicitly excluded the feasibility study and the requirements analysis from the life cycle.

The model equations

The man-power distribution is expressed by means of a Rayleigh function:

(3.19) $Y(t) = K / t_d^2 \; t \; [exp (-t^2) - exp (2 t_d^2)]$

Where,

---

8. The Software Equation links the size of a software, the development time and the total man-power to the environmental factor.

$Y(t)$	=	is the manpower needed at a time t.
t	=	the elapsed time in years or months from the beginning of the detailed logic design and coding phase. It is an independent variable representing any point in the life cycle - current elapsed time.
K	=	is the total life cycle effort in PM. It is the area under the curve Y from t= zero to infinity.
t_d	=	time (in months) at which the software development team size peaks (to a first approximation t_d is the development time for the software task, from specification to test.)

In probability theory, a Rayleigh density function is of the form:

$$(3.18) \qquad Y' = 2Kat\,[\exp(-at^2)]$$

The coefficient a determines the month in which the manpower utilisation reaches a peak. The time at which peak effort occurs can be derived as follows:

$$(3.20) \qquad t_d = (1/2a)^{1/2}$$

$$(3.21) \qquad a = 1/2\,t_d^2$$

Where,

t_d = time at which Y' is a maximum.

The shape of this curve defines the rate at which effort is consumed by the project throughout its life cycle, as shown in Figure 3.2. The model has, therefore, two fundamental parameters, namely: the cumulative life cycle effort K and the development time t_d.

project throughout its life cycle, as shown in Figure 3.2. The model has, therefore, two fundamental parameters, namely: the cumulative life cycle effort K and the development time t_d.

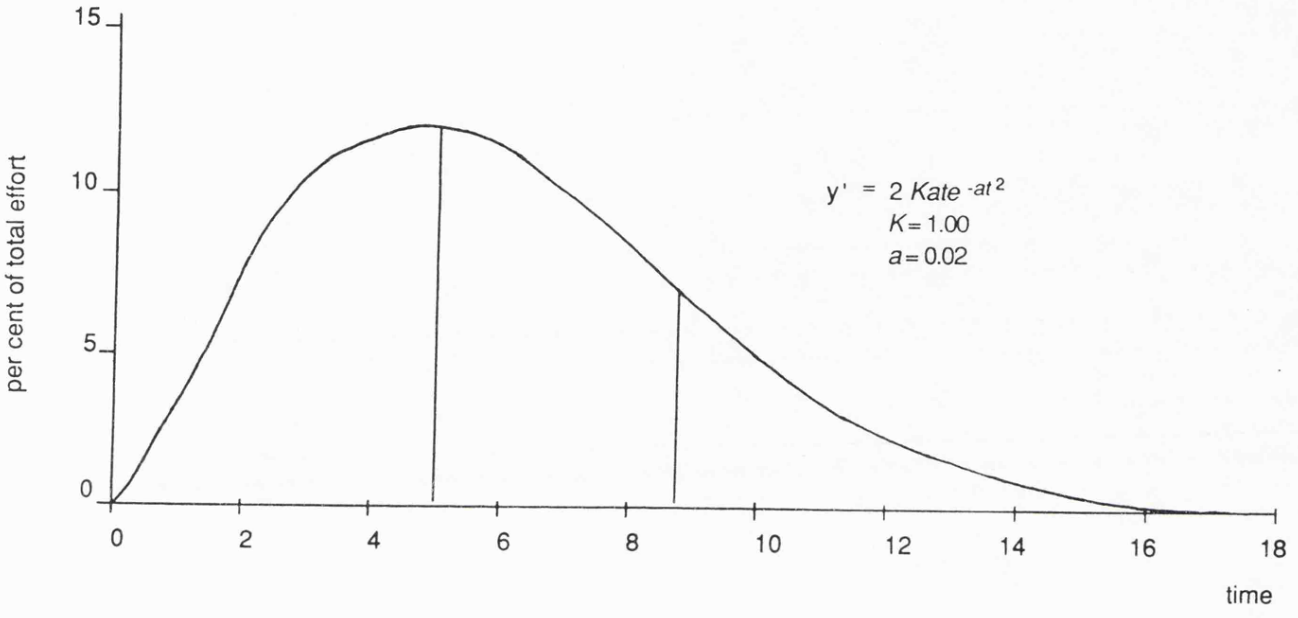


Figure 3.2 Current manpower utilisation [Nor63]

In solving equation *3.18*, it is assumed that the effort required for the software development was already estimated, and our problem is how to staff the project throughout its life cycle in the optimal (natural) way. Thus, integrating *(3.18)* over the interval [0, t], we obtain:

$$(3.22) \qquad Y = K[\,1 - \exp(-a\,t^2)\,]$$

Where,

Y = cumulative number of people used at any time t.

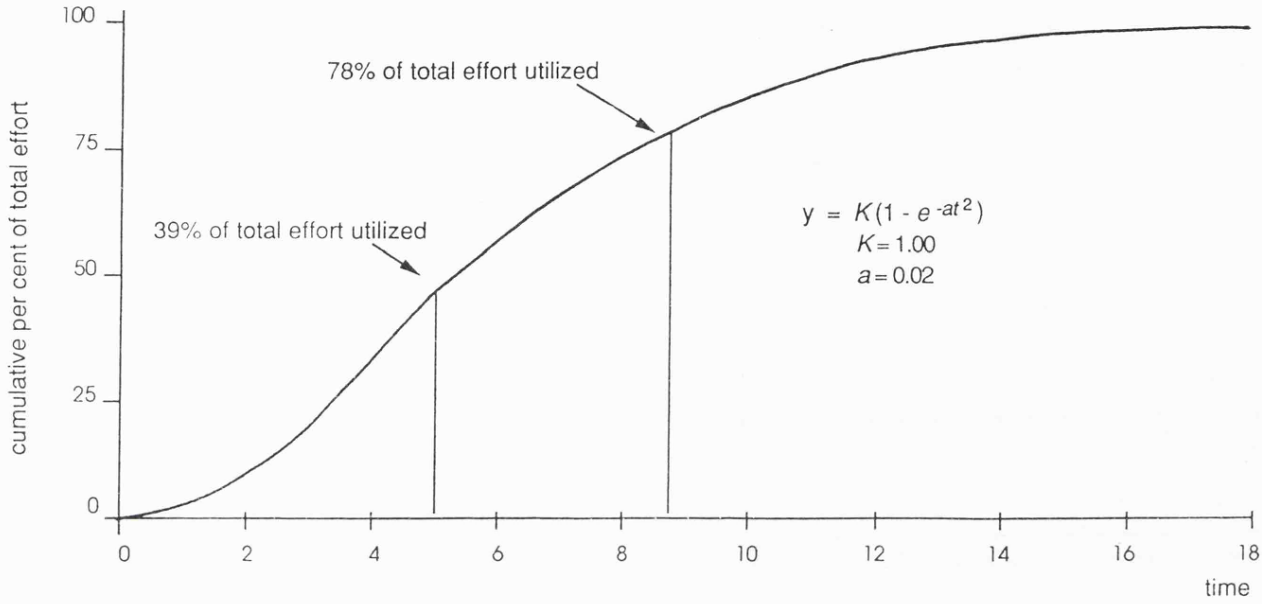


Figure 3.3 Cumulative manpower utilisation [Nor63]

The shape of the curve implies that the people involved in this type of project are learning and gaining in effectiveness. Each such curve represents a manpower staffing strategy that does not violate any natural rule about how people in a project interact. This current manpower utilisation curve has a point of inflection at a point at which the decrease in manpower utilised monthly slows down in the descending portion of the curve.

So, if we can know the month in which manpower utilisation has reached or will reach a peak, and if we know or can estimate the manpower level in that month, we can calculate the value of *K*, the total manpower required in the cycle, by substituting for *a*, as determined above:

$$(3.23) \qquad K = e^{1/2} t_d Y_{max}$$

In terms of software projects, t_d has been shown empirically to correspond very closely to the design time [Put78]. This agrees with Aron's findings as discussed in paragraph 3.2.2.

If these equations are applied only to the design and coding stage of a computer system, then:

(3.24) $Y_1 = 2 K_1 a t \ [exp (-at^2)]$

Where,

Y_1 = design and coding effort at time t
K_1 = total design and coding effort

This makes the assumption that the general shape of the Norden/Rayleigh curve applies to both the total system life cycle and also the design and coding stages of that life cycle.

The Software Equation

The relationship which links the size of the software S_s, the development time t_d, and the total man-power K to the Environmental Technology Factor C_k, is known as the 'Software Equation'.

(3.25) $S_s = C_k \ K^{1/3} \ t_d^{4/3}$

Where,

S_s = number of delivered source instructions
C_k = a technology factor
K = total life cycle effort in years excluding requirement analysis and specification, but including maintenance
t_d = development time in years

C_k, sometimes called the 'Technology Factor', assumes the technical

C_k, sometimes called the 'Technology Factor', assumes the technical appropriateness of the programming support environment, project team composition, hardware constraints and program complexity. C_k is one of 20 values in the range of 610 to 57,310.[9] It is claimed that this factor can be determined for a individual development environment from the data of past projects.

Putnam realised that Norden's findings suggested that software projects follow a life cycle curve, which is composed of a set of sub life cycle curves and that such a curve can be helpful in forming a basis on which to plan and control a software system project. This concept affords a dynamic approach to the effort estimation. The parameters K and t_d are called the management parameters. Changes in K or in t_d or both will result in a change in the shape and magnitude of the curve as shown in Figure 3.4.

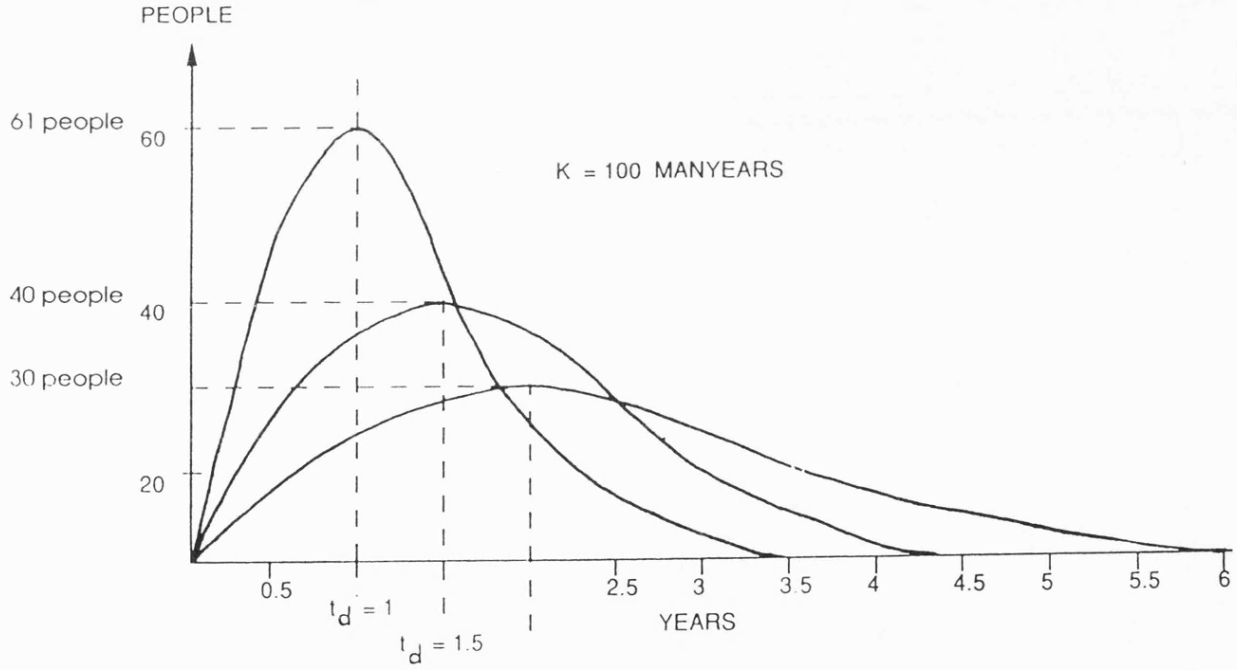


Figure 3.4 Alternative manpower loading strategies

---

9. C_k = 2,000 for a poor software development environment which do not incorporate the use of method for the development process, batch process, etc. C_k = 8,000 for a good software development environment and C_k = 11,000 for an excellent environment which involves automated tools and techniques

Difficulty Gradient

Putnam found that the ratio (K/t_d^2), has an interesting value as it represents the difficulty of the system from the programming viewpoint. Projects that exhibit a high productivity had a relatively slow initial team build-up and projects that exhibit a low productivity have a relatively fast initial team build-up. If the number (K/t_d^2) is small, it corresponds to easy systems, while a large volume for (K/t_d^2) corresponds to difficult systems. Putnam titled this ratio as the Difficulty Gradient.

(3.26) $D_i = K/t_d^3$

Where,

D_i = constant for a particular class of project.

Each such line presents the maximum difficulty gradient line that a software organisation is capable of accomplishing. That is, as system size increases the development line will also increase [Put78]. Therefore, if a system is:

- Entirely new, has many interfaces with other systems, then the Difficulty Gradient will be $D_1 = 8$;
- New and stand-alone then $D_2 = 15$;
- Rebuild or composite build-up from existing system then $D_3 = 27$;

These values vary slightly between software houses as analysed by Putnam. They are in a sense 'learning curves'.

Using this formula illustrates how management considerations can influence the project difficulty. For example, in the case that the required effort for a given project is 100 person-years and a development time of two years then the difficulty ratio is $100/2^2 = 25$. However, if a management decision is taken to reduce the project effort by 10 percent to 90 person-years, then the new difficulty ratio will be: $90/2^2 = 22.5$. Clearly, there is an assumption that the software product will have less functionality or else the difficulty ratio makes no sense.

Discussion

Feasible Effort-Time Region / The Rayleigh distribution

Putnam advocates the usage of the Rayleigh distribution for estimating the software effort on the following grounds:

- Development time for the vast majority of large systems ranges from two to five years. Five years is the limiting point from an economic viewpoint. Two years is the lower limit as result of the manpower build-up. This springs from Vysottsky's observation as cited by Brooks [Bro75] that software projects cannot stand more than 30% per year build-up. The Rayleigh equation meeting this criterion is $t_d >= 2$ years.
- The manpower rate invokes the inter communication law. Complexity = N[(N-1)/2] where N is number of people that have to intercommunicate. As the number of people on a project increases arithmetically, the number of human interactions increase in a non-linear way.
- Management cannot control the people on a large software project at rates of $t_d < 2$ years without exercising significant difficulties. As the development time is shortened , the difficulty increases dramatically.

There is support for these assumptions among researches and practitioners who advocated the evolutionary delivery for software development, which concentrates on short term results. *"There is a narrow six to twelve months 'time window' for optimum manageability"* [Gil87]. Duration, expectation and volatility are the three vital factors that impact manageability.

But, does the Rayleigh distribution correctly echo the empirical observations of the manpower behaviour in software development? Other authors have found some empirical support for the Rayleigh curve shape for very large projects, Mapp [Map78]; Parr [Par80]; Basili & Beane [Bas81]. Basili and Zelkovitz [Bas78] in an earlier study felt it was inappropriate for medium to large projects. Boehm does not agree with using the Rayleigh curve for all project types.

He concludes the shape of the Rayleigh distribution is not a close approximation to that of labour distribution curve for any of his Organic mode projects.[10] He attributes this to the fact that the project team build-up is much slower in this type of projects than assumed by the Rayleigh distribution. The team involved in this type of project is usually an in-house team (by definition of the mode type) and the project starts with a good knowledge of the application area. The central portion of the Rayleigh distribution, however, does provides a good approximation to the labour curve of the Organic mode software projects. Boehm argues that the Rayleigh distribution is a reasonably good fit for portions of the manpower distribution, particularly for the Semidetached mode, except for its zero level behaviour at the start of the project. Therefore, Boehm points out the need to tailor a portion of a Rayleigh curve to a particular mode and a particular portion of the development cycle. Kitchenham and Taylor [Kit85] found that the Rayleigh distribution did not fit the size of project and environment they analysed (mainly small projects). Warburton [War83], who analysed real-time projects, found the Rayleigh curve to be appropriate when 40% of the development effort had been used. Parr [Par80] challenges the appropriateness of the Putnam's basic assumption. He argues that the parameter a (shape of the Rayleigh curve) should not include the skills' availability for a particular project, since, these skills are constraints imposed on the project by management. Parr finds it conflicting with the intrinsic constraints on the rate at which software can be developed as discussed earlier (in the Feasible Effort Time). His interpretation is that the manpower build-up is governed by the relationships and dependencies between the problems in the project.

Kitchenham and Taylor [Kit85] found the SLIM model lacking in the following aspects:

- In practical terms, t_d is difficult to define. They found it difficult to identify the Rayleigh curve with phases in the development process.
- The accumulated life cycle effort was found to have two peaks and not one.

---

10. See Paragraph 3.5.1, for Boehm's definition of projects modes.

This finding might result from a degree of difficulty which is different from that anticipated, and thus caused a change in the PM allocated to the project.

- The choice of the technology factor which dominates the Rayleigh curve shape is difficult to make.

Effort - Time Relationships

The model raises the essential question of what the relationships are between the effort required for the development process and its duration. If we accept Putnam's Effort - Time relationship, we should expect substantial penalties for compressing the duration of development, as depicted in Figure 3.4. For example, a project requiring 20 person years of effort in two elapsed time years, would require 320 person years of effort in one elapsed time year or alternatively about 4 person years in effort in three years elapsed time.

Indeed, other researchers defined these relationships differently. Jensen [Jen 83;83a;84;86] proposed a theoretical model which is very similar to Putnam's model. A range of multipliers similar to Boehm's are used together with a curve similar to the Rayleigh curve for relating changes in time scales to product cost. However, the constant used in the equation are less extreme than Putnam's.

Jensen suggested the following relationships:

(3.27) $S_s = C_{te} T K^{1/2}$

Where,

S_s = Lines Of Code given in thousands.
C_{te} = effective technology constant.
T = elapsed time.
K = Life cycle effort.

Where,

$C_{te} = C_{tb} f_i$

And,

C_{tb} = basic technology constant.

f_i = the measure of the $_i$ environmental adjustment factor.

The f_i are similar to those processed by Boehm [Boe81].

The minimum time constraints are:

(3.28) $K = S_s^2 / C_{te}^2 T^2$

Putnam's model is based on a plausible mathematical relationship between project team size and the phase of software development. Some researchers argue that this relationship is less plausible for maintenance and a new version of existing software [Wei84].

Shooman [Sho79] suggests that Putnam's model is valid for most software development types of projects. The management parameters are appreciated as good indicators as to how cost allocation between development phases should be done. Shooman indicates that for the purpose of total cost estimation (K is known) only one or two data points for manpower levels should be sufficient to obtain a reasonably accurate estimate of the parameter t_d. Knowing t_d, we can independently calculate the significant milestones of the project, and check these results with the existing schedule.

The model suggests a way to estimate the effort and a natural staff build-up, provided that we can quantify each of the independent parameters in the Rayleigh formulae (S_s must be estimated, D_i must be known and C must be derived). None of these is a trivial task.

Putnam's assumes that the 'cost drivers' attributes are applied uniformly across the entire SDLC. This approach is adequate for coarse early estimates only.

The model has became well known and it is used by many US governmental agencies as well as in Europe and the rest of the world.

3.5 COMPOSITE MODELS

3.5.1 Boehm's COCOMO Models

In 1981 a major cost estimation model was introduced, by B. Boehm [Boe81], the COCOMO (COnstructive COst MOdel) Model. Boehm's analysed data from 63 completed projects was fitted into a pair of non-linear equations that can be used to apply this experience to new projects. The database includes projects that were developed in the period 1964-1979; they vary in size from 20K-1,000K LOC and were developed in a variety of languages. The uniqueness of this work is its comprehensiveness and perhaps more important, its public accessibility in an easy to understand form.

Boehm proposed a set of three models, each of these models with three modes of development. The complexity of the model increases in three levels along with the stated accuracy obtained in estimates using the model as follows:

* Basic-COCOMO model.
* Intermediate -COCOMO model.
* Detailed-COCOMO model.

The three modes of development domain classified by Boehm are:

* The Organic mode.
* The Embedded mode.
* The Semidetached mode.

They are defined by product type, certain characteristics of the project and its team composition, namely:

- The **Organic** development mode is characterised by small teams, working in a familiar in-house environment. Thus, the initial learning and

communication load is not high and there is room for negotiation when difficulties arise.

- The **Embedded** development mode is characterised by relatively large projects, complex operating environments which spring-up from complex hardware software and operational inter-relationships. The project team is not familiar with similar projects and there is a high initial learning and communication load. Thus, there is a slow project start as technical interfaces have to be resolved. There is some freedom to negotiate requirements.
- The **Semidetached** development mode falls somewhere between the Organic and the Embedded Modes.

The models equations and assumptions

As in the Putnam model there are underlying equations describing relationships between effort, size and elapsed time. These relationships are non-linear and take the functional form:

(3.29) $Effort = a * Size^b$

Where,

Effort = number of PM (a month = 152 working hours).

Size = measured in thousands of Delivered Source Instruction. Delivered Source Instruction (DSI) are defined as program instructions created by project personnel that are delivered as a part of the project. Comments and unmodified utility software are excluded.

(3.30) $Elapsed\ time = c * Effort^d$

Where,

Elapsed time describes how much time it takes to use a number of person-months of effort effectively for a typical project.

The values of the model parameters *a, b* and *d* are dependent on the mode of development and the level of COCOMO model in use. The parameter *c* is dependent on the mode of development only. Thus, the elapsed time varies with the mode of development.

The following equations define the productivity and the average staffing:

(3.31) *Productivity* = *Size/Effort*

(3.32) *Average (FSP) staffing* = *Effort / Elapsed time.*

Where,

FSP = the full time equivalent software personnel.

The Basic COCOMO is a relatively simple model which aims at coarse and quick estimates. Only one primary parameter affects effort, the size measured in DSI. Yet, the assessment of the project's mode defines the appropriate coefficient (e.g for the Organic = 2.4; for the Semidetached = 3.0 and for the embedded = 3.6). However, the Intermediate and Detailed COCOMO models are more sophisticated versions. The assumption is that it is harder to produce DSI under very high reliability requirements, time and storage constraints. Boehm introduces fifteen independent variables which are attributes of the end products, which he believes influence the value of the equation's parameters and therefore, are suitable for refining the equation. Boehm groups these factors, which he calls cost drivers, into four categories, namely: product attributes; computer attributes; personnel attributes and project attributes. The cost drivers and their rating is given in Table 3.2.

	Rating					
	Very low	Low	nom-nal	High	Very high	Extreme high
Product attributes.						
- required reliability.	.75	.88	1	1.15	1.4	-
- database size.	-	.94	1	1.08	-	-
- software product complexity.	.70	.85	1	1.15	1.30	1.65
Computer attributes.						
- execution time constraint.	-	-	1	1.11	1.3	1.66
- main storage constraint -	-	1		1.06	1.21	1.56
- virtual machine volatility	-	.87	1	1.15	1.3	-
- computer turnaround time	-	.87	1	1.07	1.15	-
Personnel attributes.						
- analyst capability.	1.46	1.19	1	.86	.71	-
- applications experience.	1.29	1.13	1	.91	.82	-
- programmer capability.	1.42	1.17	1	.86	.70	-
- virtual machine experience.	1.21	1.10	1		.90	
- Programming language experience.	1.14	1.07	1		.95	
Project attributes.						
- modern programming practices.	1.24	1.1	1		.91	.82
- use of software tools.	1.24	1.1	1		.91	.83
- schedule constraints.	1.23	1.08	1	1.04	1.10	-

Table 3.2 The Cost Drivers affect the size/effort tradeoff [Boe81].

The basic equation then takes the form:

(3.33) $Effort = a * Size^b * M(X_i)$

Where,

$M(X)$ = a correction factor, for the cost drivers effect on the specific project. Each cost driver attribute has a set of multipliers which are keyed to a set of project rating for the attribute. $M(X_i)$ is the product of all fifteen cost driver's multipliers, which is then multiplied the nominal effort.[11]

$M(X)$ = 1, if all cost drivers are given a nominal rating.

A development project is divided into the four following phases:

* Product Design.
* Detailed Design.
* Coding and Unit Test.
* Integration Test.

The cost drivers are estimated and applied to each phase separately. The Detailed Model assumes that the influence of the cost drivers is phase dependent, while Basic - Organic and the Intermediate - COCOMO models do not. These two models distinguish only between development and maintenance. Boehm recommended that when using the Detailed model some of the cost drivers be applied at a module level, some at a subsystem level and some at the system level. The coefficients for these equations corresponding to the various modes of development are given in Table 3.3. To obtain estimates of effort, the size of the

---

11. For example, 'use software tool', which is one of the cost drivers, has five numerical values corresponding to the five possible ratings (as shown in Table 3.2). Then, the effort (PM) is multiplied by the numerical values corresponding the selected rating.

product must be estimated and the mode of development must be specified.

	Basic COCOMO				Intermediate COCOMO			
	Effort		Schedule		Nominal effort		Schedule	
	a	b	c	d	a	b	c	d
Organic	2.4	1.05	2.5	0.38	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32	2.8	1.20	2.5	0.32

Table 3.3 The basic effort and schedule coefficients for the Basic and Intermediate types of projects [Boe81].

Methodology

The equations were obtained using a combination of experience, results of other cost estimation models, the subjective opinion of experienced software managers and trial-and error to arrive at initial model parameters based on a subset of the entire database. These initial parameter values were further refined, tuned and calibrated using additional projects from the database.

Project Profile and Resource allocation among phases of development

Boehm defines a project profile as a function of project size. It can be represented accordingly by manipulating the COCOMO equations, as a function of effort, schedule, average staffing and productivity rate. Effort and schedule distribution among phases of development is suggested to be a function of the product size. Even though, both large and small Organic projects have relatively flat labour distributions compared with other modes of development. The phase distribution for effort and schedule for the various models are given in Table 3.4 and the project classification as function of project size is shown in Table 3.5.

Project size	Project profile and effort allocation (percentage) 2K	8K	32K	128K	512K
EFFORT					
The Organic Mode					
Plan & requirements	6	6	6	6	
Product design	16	16	16	16	
Programming	68	65	62	59	
Integration & test	16	19	22	25	
The Semidetached Mode					
Plan & requirements	7	7	7	7	7
Product design	17	17	17	17	17
Programming	64	61	58	55	52
Integration & test	19	22	25	28	31
The Embedded Mode					
Plan & requirements	8	8	8	8	8
Product design	18	18	18	18	18
Programming	60	57	54	51	48
Integration & test	22	25	28	31	34
SCHEDULE					
The Organic Mode					
Plan & requirements	10	11	12	13	
Product design	19	19	19	19	
Programming	63	59	55	51	
Integration & test	18	22	26	30	
The Semidetached Mode					
Plan & requirements	16	18	20	22	24
Product design	24	25	26	27	28
Programming	56	52	48	44	40
Integration & test	20	23	26	29	32
The Embedded Mode					
Plan & requirements	24	28	32	36	40
Product design	30	32	34	36	38
Programming	48	44	40	36	32
Integration & test	22	24	26	28	30

Table 3.4 Phase distribution of effort and schedule [Boe81].

Boehm provides us with an algorithm to deal with values which fall in between the interval of the discrete values presented in Table 3.5.

Size of project	Organic			Semidetached			Embedded		
	PM	SCHE-DULE	ESP	PM	SCHE-DULE	ESP	PM	SCHE-DULE	ESP
Small, 2KDSI:	5.0	4.6	1.1	6.5	4.8	1.4	8.3	4.9	1.7
Intermediate 8KDSI:	21.3	8	2.7	31	8.3	3.7	44	8.4	5.2
Medium, 32KDSI:	91	14	6.5	146	14	10	230	14	16
Large, 128KDSI:	392	24	16	687	24	29	1216	24	51
Very large, 512KDSI:	-	-	-	250	42	77	6420	41	157

Table 3.5 Project classification as function of project size [Boe81].

By linear interpolation and by using the observed distribution of effort between phases of development, the effort and schedule for project of particular size can be calculated.

Kitchenham & Taylor [Kit85] conclude with a similar average phase distribution while comparing COCOMO in the ICL and British Telecom (BT) environments. Yet, individual projects vary widely from this average.

Discussion

The basic set of equations, their associated coefficients, project classification, the 'standard' effort and schedule allocation among phases provide a vehicle for the planner of a software project development, given that it is feasible to estimate the project size and characterise the development mode. Although a given project may deviate from the pattern suggested by the elapsed time equation, it makes good sense to plan on delivery time consistent with this equation.

Boehm explicitly incorporates into his models the additional independent variables, the cost drivers (while Putnam includes them within the constant C_k).

The introduction of the cost drivers to the Intermediate and the Detailed models allow us to adjust the estimated effort to individual characteristics of a project. The values of the cost drivers afford us to crystallise the role played by each aspect of software development and to act to improve the situation. This could be used as a powerful management instrument. Moreover, if we use the models after major software components are identified, we are able to tune the estimate by applying the cost drivers to the individual identified component. The intermediate and the detailed model present an increasing degree of sophistication to the estimation process. The detailed model is a phase sensitive model, in which the cost drivers applied differently to each of the phases and or segments of development. This helps in the manning the particular phase and/or segment. These powerful features are attributed to the 'cost drivers' under the assumption that they add information.[12] However, most of the attributes on this list were covered by his fellow researchers.

As noted earlier, the COCOMO models do not deal with the planning and the requirements phases. They are targeted towards estimating the programming effort and, thus, miss some of the important attributes that have impact on the early stages of the system development such as the organisational factors mentioned by Walston and Felix [Wal77], Aron [Aro69], Nelson [Nel66] and Doty [Dot77].

An additional feature of COCOMO models is the inclusion of instructions for adjustment of the LOC to reflect the use of existing software which is adapted for current projects. The model suggests algorithms for handling re-useable of code and assessing maintenance. Recently, the original model was expanded to include risk assessment and modern process model, this is the Spiral SDLC [Boe88]).

Although the model presents a hierarchy of models, which provide a way of estimating the effort at the different levels of understanding the project under development, this is a typical micro model. It is the bottom-up estimation process

---

12. This assumption was questioned by the Kemerer study [Kem87]. The results of his study do not support this statement.

which gives us insight into factors affecting the process of software development and, hence, helps in understanding the effort estimation process.

It should be noted that the models are suitable and intended for use after requirements are completed. However, the estimated effort includes the effort for the requirements and the project planning, using resource allocation among the stages. The model covers the management and documentation efforts but excludes some efforts which take place during the development period, such as user training, installation planning and conversion planning.

Theoretical issues, methodology

Conte [Con86] has criticised the COCOMO model set on a number of fronts:

- Too many parameters. Boehm uses 15 parameters in addition to project size. The range of these 15 attributes can result in a wide range of estimated effort. $M(X) = 72$ if each cost driver is assigned the highest attribute and $M(X) = .088$, if each cost driver assigned the lowest attribute.
- The cost drivers' attributes and the constants derived empirically, needed tuning to fit the database. It is therefore questionable whether the model has the quality of a universal model.
- The quantification of the attribute to two significant digits is questionable.

Empirical issues

Although theoretically desirable, the quality of a model should not only be judged by the number of parameters used. COCOMO models use 15 attributes to tune the nominal effort equations to the individual project. However, the estimator knows what the correction factors are and, thus, is able to analyse their effects and to calibrate them accordingly. This is no less legitimate than using a model which incorporates only a few parameters which are a 'black box' to the estimator. In such cases the estimator does not know how to obtain the parameters included in the 'black box', nor does he understand how his input is interpreted to form the

parameters.

Empirical research indicates poor results, mainly for the Intermediate and the Detailed model. Kitchenham and Taylor [Kit84;85] who analysed data from ICL and British Telecom (BT), conclude this as a poor fit. Kemerer [Kem87] who evaluated Putnam's SLIM, Boehm's COCOMO and Rubin's ESTIMACS concludes similarly for all three COCOMO models. Kemerer points out that the average error for all versions of the model was 601%, with the lowest single error being 83%.[13]

COCOMO models are considered by many to be the most applicable to other environments. Cowderoy [Cow86] indicates (from discussions at meetings of the user community of COCOMO) that the company with a better tailored version of COCOMO has the advantage in tendering for the DOD contracts.

3.6 COMPARISON AMONG MODELS

3.6.1 Economies and diseconomies of scale

A development process presents 'economies of scale' when the average productivity is increasing (the marginal return of an additional unit of input exceeds the average return), and 'diseconomies of scale' when the average productivity is decreasing.[14] Boehm [Boe81] concludes that 'diseconomies of scale' prevail for large projects. Boehm's conclusion agrees with those of many other researchers such as Brooks [Bro75], Aron [Aro69], Putnam [Put78;79] and Wolverton [Wol74]. From Table 3.6 (adopted from Boehm [Boe81]), we can

---

13. Kemerer suggests that these results may be due to the TRW data used for the development of the models. This data vary in size of the projects and their composition form the data used as source for this experiment.

14. When a 'diseconomy of scale' exists, the effort equation exhibits an exponent which is greater than one and the productivity rate decreases.

observe that the only two models that do not suggest 'diseconomies of scale' are the Walston and Felix [Wal77] and the Nelson [Nel66] models.

Walston	$5.2^{0.91}$
Nelson	$4.9^{0.98}$
Freburger	$1.48^{1.02}$
COCOMO - Organic	$2.4^{1.05}$
Herd	$5.3^{1.06}$
COCOMO - Semidetached	$3^{1.12}$
Fredric	$2.43^{1.18}$
COCOMO - Embedded	$3.6^{1.20}$
Phister	$0.99^{1.275}$
Jones	$1^{1.40}$
Walston	$1.12^{1.43}$
Halstead	$0.70^{1.50}$
Schneider[15]	$28^{1.83}$

Table 3.6 Comparison of effort equations [Boe81]

Banker and Kemerer [Ban88] did not accept the harmony observed in earlier studies, as shown in Table 3.6. They suggest that the effort required for software development may be either 'economy or diseconomy of scale' and argue that a locally increasing or decreasing return to scale depends upon the size of projects. Banker and Kemerer came to this conclusion after analysing data, from various sources, representing a variety of applications. Increasing returns to scale were observed for the:[16]

---

15. This is not a mistake!

16. Yourdon's data: various business application, 22 projects) [DeM82]; Baileys' data: NASA, 19 projects [Bai81]; Behren's data: Equitable Life Assurance Society, 25 projects [Beh83]; Kemerer's data: Commercial data processing, using Function Point; [Kem87].

Yourdon	an exponents of .72
Bailey	an exponents of .95
Behren	an exponents of .95
Kemerer	an exponents of .85

And, decreasing returns to scale for:[17]

Albrecht	an exponents of 1.49
Belady and Lehman's	an exponents of 1.06
Wingfield's	an exponents of 1.06

The authors associate this phenomenon with project size. They suggest that the software development first exhibits increasing returns to scale in most organisations, but decreasing returns for very large projects. Whether a model results in 'diseconomy or economy of scale' depends on the complexity of the project itself, the technological environment, the project team composition and or the organisational factors [Ban88]; [Boe81]; [Bro75]. Software development tools such as on-line debuggers or code generators should increase productivity. However, although modern tools might increase productivity level in the long term, they require relatively large initial investments in purchasing and adapting to the organisation. The factors which may contribute to 'economies of scale' are:

- Management overhead which does not increase directly with project size.
- Software development tools if used in an organisation repeatedly despite the initial investment in purchasing and learning curve.

The factors which may contribute to 'diseconomies of scale' are:

- Size of project team might have conflicting effects. The size of the project

---

17. Albrecht's: IBM, 24 projects [Alb83]; Belady and Lehman's: software house, 33 projects [Bel79b]; Wingfield's: US Army, 15 projects estimated using SLIM [Win82].

team affects a number of communication paths between members of the project team. The number of communication paths increases in a non-linear rate. In addition, personal conflicts may increase with size of team. However, size of the project team might be a cause for 'economy of scale'. For instance, large project team may benefit from the accessibility to high level specialised personnel whose expertise may increase the overall productivity of the project [Ban88].

- Overhead activities such as planning and documentation grow faster than at a linear rate. It is customary to assume that a large project will end with a more complex interfaces than a small project. The need for parallel activities exists in large projects. A large project requires relatively more time to implement the integration and test activities. Verifying, validating, testing, integration and managing the project are the activities which are mostly affected by the project size as relatively more time is needed for communicating and resolving interface problems.

It would be of benefit to the software industry if a way to size a new software development project which would reflect 'economy of scale' could be suggested. But, this problem does not concern us here.[18]

---

18. Banker and Kemerer address this issue and suggest a research direction which might allow for the identification of the scale size for software project development where average productivity is maximised [Ban88]. The authors suggest a method for identifying the most productive scale size for a given software development environment. They argue that most small projects exhibit a increasing return to scale while vary large projects decrease in their productivity and exhibit 'diseconomies of scale'. That is, average productivity is increasing as long as the project size is smaller than the 'Most Productive Scale Size' (MPSS) and is decreasing for projects that are larger. The MPSS exists where the marginal productivity equals the average productivity. The actual MPSS tends to differ between organisations. In other words what they are suggesting is that for a given team there is an optimal project size. Cowderoy, Jenkins and Harry [Cow88b] suggested that a more sensible approach is to look at the optimal staffing pattern for a project of a given size.

3.6.2 Comparison among schedules

The effort and schedule equations obtained in various models are difficult to compare as assumptions and definitions are not always available. However, one can realise the remarkable agreement shown in the various scheduling coefficients and constants, as shown in Table 3.7. *"Even though, to date no one has came up with a good explanation for this relation in terms of project phenomena"* [Con86].

Freburger - Basili	$4.38^{0.25}$
COCOMO - Embedded	$2.5^{0.32}$
Putnam 78	$2.15^{0.333}$
COCOMO - Semidetached	$2.5^{0.35}$
Walston 77	$0.247^{0.35}$
Nelson	$3.04^{0.36}$
COCOMO - Organic	$2.5^{0.38}$

Table 3.7 Comparison of schedule equations [Boe81]

3.6.3 Sensitivity to elapsed time

Boehm's COCOMO and Putnam's SLIM are sensitive to compression in elapsed development time, as is PRICE-S [Frei79], the Jensen model [Jen84] and the BANG model [DeM82]. They are all based on the assumption that there is a relationship between the elapsed time and the effort required to develop a software product. Reducing the time frame may increase total effort by making co-ordination more complex and reducing individual efficiency. The theory states that there is a cost involved in shortening the elapsed time in a way that conflicts with the 'natural' schedule for a project of a given size [Put78]; [Boe81]; [Wol84]. Extending the time frame may also reduce the momentum in part of the team and extend duration on project management. Putnam titled this as the Time Trade-off Law and suggested that there is an optimum elapsed time for a project in which

the required effort to develop the system is minimised.[19]

3.6.4 Comparison: Putnam's SLIM and Boehm's COCOMO

SLIM and COCOMO models are archetypal effort estimation models. They differ in their assumptions and their interpretation of the observed behaviour of effort throughout the SDLC. It is therefore of benefit to compare them.

Similarities

The similarities between Putnam's and Boehm's models are:

- The functional form of the equations; both are non-linear.
- The basic variables in the equation are size, effort and schedule constraints. Both models specify the relationships among the various cost parameters e.g. effort/cost and duration/staff.
- Both models interpolate the total development effort and schedule, and from that, based on resource allocation among the phases of development, estimate the effort for the construction phase of the project.

Some authors argue that no one model adequately represents all task types and environmental factors in a totally convincing fashion. For instance, Basili [Bas81b;81c], Pressman [Pre87] and, Macro and Buxton [Mac87] who suggested that *"No one of the models developed is conclusively better than all others in all circumstances"*. Putnam and Boehm take a different view. Both suggest the use of a single model, adjusted for usage in various environments. A similar view is taken by Rubin [Rub85]. He suggested a single coherent model that can capture and make use of the range of estimation parameters when known.

---

19. This subject was also explored by [Cow88b], who tried to identify the optimised 'cost' or 'elapsed time'.

Differences

Elapsed time constraints. The two models vary in their explanation of the effect of the elapsed time constraints and expansion, as shown in Figure 3.5.

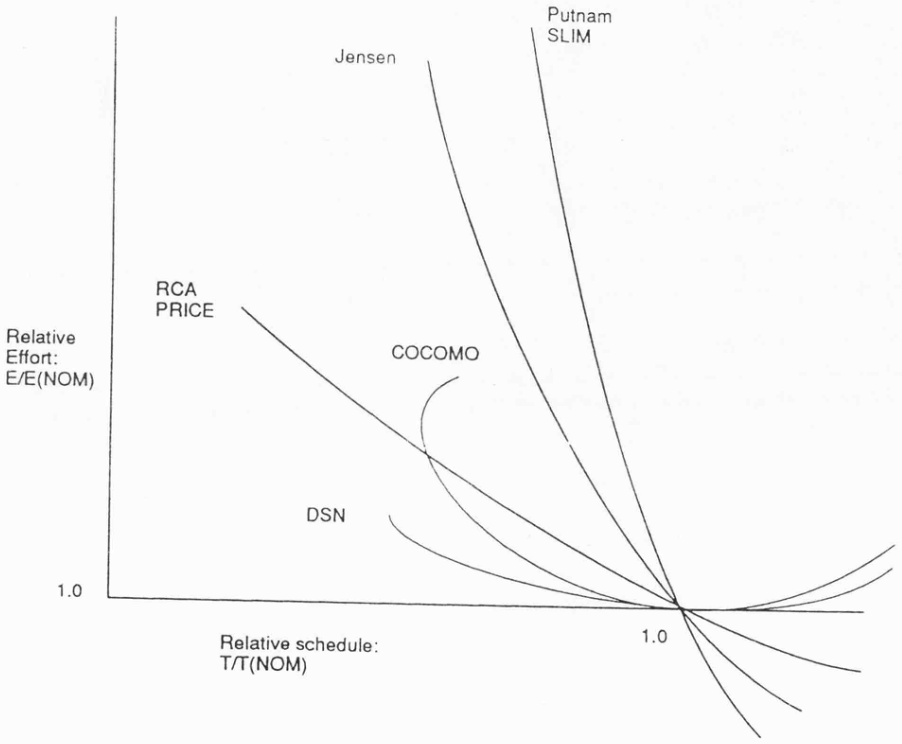


Figure 3.5 Relative effort and elapsed time [Mac87].

This figure shows the relative elapsed time on the x-axis and relative effort on the y-axis.[20] Thus relative effort shows the extent to which effort is more or less than would be expected for a project of a certain size, assuming no elapsed time or

---

20. Relative effort (E/E_{exp}) is calculated by dividing a project's actual effort (E) by its expected effort (E_{exp}). Expected effort is calculated from the empirically based equation describing the relationship between lines of code and effort $(E_{exp} = a * lines\ of\ code^{b})$.

schedule pressures. Relative elapsed time measures the extent of schedule constraints by dividing the desired completion time, which is enforced on the particular project, by the elapsed time that would be normal, based on the empirical equation describing the relationship between actual effort and actual elapsed time *(elapsed time = c * effort^d)*.

Boehm believes in the existence of an 'impossible region' for development, which represents infeasible manpower strategies. Acceleration of the nominal elapsed time of a project as given in equation *(3.30)*, below 75% of the development time for a project as given in equation *(3.29)*, is considered as impossible by COCOMO.

Putnam [Put78] gives the following relationships for the minimum development time T_d for a stand alone project:

(3.35) $K = C\,T_d^3$

Where,

K = total effort in Person Years.
T_d = measured in years.
C = in the range from 14 to 15.

In the case C equals to 14.5, then converting to months:

(3.36) $T_d = 2.15\,(PM)^{1/3}$

C = 2.15 in the Putnam equation represents a typical minimum compression of schedule of about 86%, while in COCOMO models C = 2.5 which represents only 75% schedule compression limit. Putnam's effort-schedule tradeoff equation *(3.36)* presents an extremely steep penalty for compression in schedule, and an extremely reduced effort for spread of schedule.

There is clearly a limit to the extent to which effort can be squeezed within a given elapsed time. DeMarco agrees and describes the phenomenon as a *"weapon in your arsenal to use against unreasonably inflated expectations dropped on you from above"* [DeM82].

3.7 CONCLUSIONS

The selection of models analysed in this chapter aimed to give the reader an understanding of the modelling approaches for estimating the effort for software development. Thus, each category[21] of the model is represented, emphasising models which contribute to the research and practice in this area of interest. The selection also includes models which illuminate aspects of productivity (e.g. [Wal77]) of software development, and models which emphasise the process associated with estimating the effort (e.g. [Aro69];[Wol74]).

A summary of the major themes of interest, resulting from the analysis presented in this chapter is given below.

The models for estimating the effort and/or cost for software development result in one or more of the following:

- **Unit of cost,** cost per instruction, cost per routine or module, cost per activity.
- **Parametric Equations,** the project required x modules, y types of data, z displays etc., and the effort or cost for the development is the summation of the products of the units of each of these modules, displays etc., with its corresponding cost parameter. Equations *(3.1)* and *(3.7)* are representative of this group.
- Some models result in a **set of parametric formulae:**

  * A **basic formula,** representing the required effort in PM or PY. The size of the software product, given in LOC, is the variable needed to solve the effort equation. The estimated effort can be adjusted.
  * A **correction formula,** accounting for a set of factors which are believed to impact the productivity of the development effort. These factors are either attributes of the software product, or of the environment in which the development occurs. Equations *(3.2)* and

---

21. See Paragraph 3.1.2

(3.3) are representative of this approach.

- Some models add a **nominal schedule** formula, representing the required schedule for the development of the software for which the effort was estimated. Equations *(3.12)* and *(3.30)*[22]. The schedule is derived based on historical data, with known percentage of resource distribution among the phases and stages of development.
- Other models **present the staff build-up**, of the effort required for the specific development, through the life cycle. The manpower demand is represented analytically by means of a Rayleigh distribution. The distribution shows how the total effort is used during the life cycle of a project, for different values of the parameter *'a'*. Restated here, is:

(3.18) $Y' = 2Kat\,[\exp(-at^2)]$

The representative of this approach in this chapter is Putnam's SLIM, based on Norden's findings. Two parameters govern the 'staff build-up' curve are:

* Cumulative effort
* Development time in which the manning of the project will reach the peak.

The month in which the manning of the project will reach the peak is related to the parameter representing the 'learning curve' associated with the project {*('a'*, in *(3.21)*}. Therefore, if *'a'* is known and constant, then the number of people involved in a project at the peak time is easy to determine.[23]

The total effort required for the development is calculated using

---

22. See also Table 3.7

23. This is achieved by replacing *'a'* with *(3.21)* in *(3.18)*.

(3.22) and assuming the duration of the total project. This model assumes the 'ideal' staffing conditions which may not be available, therefore, tradeoffs must be made.

The top-down approach is used in some models while in others the bottom-up approach is employed. However, the input required by the models requires knowledge about the software product. Estimating the LOC, which is the major input to most of the models, can be achieved only after a thorough analysis is implemented.

The correction factors used to adjust the basic formula to a specific environment are assessed subjectively and intuitively since neither the structure of the software product, nor the the correction factors for the environment are known early in software development. This implies uncertainty which is the major reason why most of the models are employed only after requirements and product specification are implemented.

All the models analysed are empirically driven. Project histories are used to establish the parameters of the formulae representing the effort, schedule and their relationships. This is true for formulae derived using statistically methods, (e.g. least squares or regression) or for those which are based on analytical methods, (e.g Rayleigh distribution).

The manner of treating the effort-time tradeoffs clearly differs among the models.

The discussion in this chapter and the analysis of the models describes indicated the importance of estimating the size of the software product. The next chapter focuses on measuring the size of the software product.

Chapter 4

ESTIMATING THE PROJECT SIZE

4.1 INTRODUCTION

Sizing the product is the weakest link in estimating the effort for software development. A weakness of all effort estimation models which base their effort estimation in LOC is that it is very difficult to estimate LOC. The clear need for better size estimation, coupled with the new technology made available to the industry this decade, introduced a new family of tools aimed at supporting the estimation of the project size. This has resulted in the development of LOC estimation tools which can be used to generate the inputs to the cost models, some of which act as front-ends to the proprietary effort estimation tools. This new family of models uses the same approaches described earlier as applicable for estimating the effort required for developing software: Analogy; Expert judgement; Delphi; and additional techniques such as Size-in-size-out and Function Point Analysis. Some of the models use Expert Systems techniques, e.g. Naef [Nae88] and Najberg [Naj88].

This chapter focuses on two problems associated with estimating the size of a software project, for the purpose of estimating the required effort:

1. Improving the estimates of LOC for the software product. Representatives of the various approaches currently in practice for this purpose will be

discussed, they are:

* Sizing by Analogy.
* Comparison of Project Attributes.
* Size-in-size-out.
* Linguistic Approach.[1]

2. Using alternative non-LOC units of measurement for the software product. These measurement need to be available early in the life cycle, and they must be amenable to counting. The various approaches currently in practice for this purpose are:

   * Function Point Analysis.
   * DeMarco's Bang.

4.1.1 Standard Measure for Unit of Product

> *"To estimate the software cost on the basis of LOC is analogous to estimating home construction cost based on the number of nails or bricks to be used"* [Cal84]

Before an attempt is made to estimate the effort required to develop a piece of software, there is a need to estimate the project size. Most of the effort models currently in use rely on an estimate of project size in LOC as a primary factor. Although the LOC, as the software product size, is paramount to effort prediction, it is very difficult, if not impossible to estimate at the outset of a project. *"It is an illusive goal"* [Con86]. A percentage degree of error in the size estimate will result in an even larger percentage error in estimated effort. Most of the effort equations are non-linear and exhibit an exponent which is greater than one:

---

1. This approach, which is based on Halstead's [Hal77] Software Science Law, will be discussed in Paragraph 5.6.4 as an approach to measure complexity.

$$Effort = a\ Size^{b},\ b > 1$$

The higher the exponent, the higher the error in the estimate. For example, the estimated effort for a project classified by Boehm as an Embedded Mode project, has the coefficients:

$$Effort = 2.8\ Size^{1.2}$$

Therefore, a 150% error in estimated size will result in a 166% error in effort estimates. As most of the models exhibit 'diseconomies of scale' and human errors are not uncommon in this area, it should not be a surprise that the corresponding error in the effort estimates is high. Even when a system matures, when the Preliminary System Design is already implemented, the requirements are specified and 'stabilised', interfaces defined and processing functions identified, the process of sizing software is still subject to a wide margins of error. LOC can be difficult to estimate for totally new projects, but, it might be even more difficult to estimate the size for projects when new code is incorporated into old software.

Many factors contribute to this phenomenon, but the unifying characteristic of them all is the potential for error inherent in any subjective human reasoning process. Some of the problems might stem from terminology and semantic definitions, some from redundancy in calculation, or design and environmental constraints such as maintenance, or high quality standard for the product (e.g. software tool), high reliability requirements, or a high degree of user friendliness. It should, therefore, be asked if the LOC qualifies as a standard unit of measurement of product. Nevertheless, the majority of productivity studies have used LOC per period of time as their productivity metric and thus the measure for a unit of software product. Johnson [Joh77] addresses the applicability of LOC and concludes that it is the *"only usable measure of program development productivity available."* Prell and Sheng [Pre84] came to the same conclusion: " *the most practical and widely used metrics are still based on the line count of software."*

Productivity in the majority of studies to-date is expressed in terms of Delivery Source Instruction (DSI) per person-day. Hence, software factors,

organisational, technical or project team composition factors are incorporated in the calculated productivity rate.[2]

(4.1) $Productivity = LOC / PM$

The uniformity of programming languages used in the early years when programs were coded mainly in one major language, the Assembly language, enabled LOC to be the standard measurement of effort, although with some difficulties. Unfortunately, today the LOC no longer qualifies as a standard measure, for the following reasons:

- Differences between individual producers of LOC can be enormous, [Bro75], [DeM82], [Boe88]. Boehm, Brown and Lipow showed a one to ten difference in error rates between personnel [Boe76a]. Differences between individual teams of producers of LOC can also be enormous, [Jon86]. In addition, variability in human ability is ignored and the person months (PM) factor assumed to be of an 'average' level of skills.
- Some LOC are more complex than others [DeM82].
- Cost-per-defect measures penalise high quality programs [Jon86].
- LOC measures penalise high-level languages and the use of advanced programming tools, e.g. reports and screen generators, program generators.

The above discussion, however, does not depict the entire picture, which is even more complex stemming from the lack of a standardised definition of LOC. Walston and Felix [Wal77] ignore object code and their report counts only source lines of code although they do include in their formulae comments that consist of up to 50% of the code. Doty [Dot77] and Boehm [Boe81] use source lines of code only and do not compensate for documentation. Yet, Boehm uses a complicated formulae that accounts for the percentage effort required for the adaptation of old code in the various SDLC phases as defined in his model (design, code and

---

2. [Wal77]; [Chr78]; [Jef79;81]; [Bai81]; [Bas81a]; [Jon86].

integration). Bailey and Basili [Bai81] define source LOC as a sum of new code + 20% of old code. Reifer [Rei88] excludes comments and data declaratives while Boehm includes the data declaratives lines. Putnam [Put79] includes overheads that account for about 50% of the code, in his S_s factor. One difficulty is very obvious. LOC is not the natural measure of productivity for the phases in the development life cycle which are not oriented towards programming. Therefore, most of the models which are based on the LOC approach exclude the early phases of the SDLC from the estimation process. Alternatively, they interpolate it from the total project size according to the industry average resource allocation between the various phases.[3]

The LOC might be a suitable measurement for the programming effort, although the LOC are not identical in the effort they consume, even when the same language is used [Dot77]. In addition, its usage as a productivity measure penalises well designed but short programs. The LOC volume is not known early in the life cycle, so it is not a suitable measure for other life cycle phases.

4.2 IMPROVING THE ESTIMATES OF LOC

4.2.1 The general approach

The new supportive tools are based on the assumption that accuracy in the LOC estimate will result in more accurate cost estimates. The appropriate analogy is selected using confidence levels. The tool proposes an analogy based on system characteristics or system functionality or both. Systems with the highest number of matching systems characteristics and matching functionality are offered to the cost analyst as the best analogies. It is claimed that the tools have the capability to translate the high level system description into a taxonomy of software

---

3. The SLIM model, the TRW Model and the Software Estimation and Evaluation Resources (SEER) System [Gal88] are examples of this approach.

characteristics. The system analogous to the target system is later scanned for detailed data. The new system size is calculated based upon the portion of the new system for which an analogy was found. A sample of the various approaches and tools aiming to support the process of estimating the software size are described and analysed in the following paragraphs.

4.2.2 Sizing by Analogy

The Sizing by Analogy approach involves comparing a new project to previously developed components of a software product, such as modules and sub-systems, and generating size estimates from the data from other similar projects. A prerequisite for using this approach is the construction of a data-base consisting of descriptions of either completed projects or ongoing projects. The data-bases usually include decomposition of the product into functions the and number of Source Lines of Code (SLOC) for each function. The analogy can be applied either at the system or at the function level, depending on the degree of detail at which the analogies are drawn. The comparison at the system level is based on comparison of project attributes, and works similar to the comparison with the cost driver, while a functional comparison works at the level of application. It is obvious that searching for analogy at the functional level may yield better results, but can be implemented only at the point in the SDLC where the system definition is completed.[4]

Electronic System Division (ESD) Sizing Model

The Electronic System Division Sizing Model consists of two primary components:

---

4. This approach is being used by the various commercial models yet in different ways. The Electronic System Division (ESD), Software Sizing Package developed in 1987, the Software Sizing Analyser (SSA) developed in 1985, the Quantitative Software Methods (QSM), are examples of tools using the Sizing by Analogy.

an historical data-base and a user interface. The user interface facilitates the use and maintenance of the data-base. It has data extraction and statistical report generation facilities.

The data-base

The sizing data-base consists of 825 (September 1987) previously designed or developed units of code which originated in four governmental and research agencies in the US. These entries are grouped and indexed into approximately 105 standard functions. The size range of the units vary from 2 to 500K SLOC. Examples of the standard functions are: On-line Monitor; Data-base design; Test case generation; Sort/Merge; Avionic navigation. An entry descriptor includes the following:

* Standard function.
* System identifier.
* Status of the unit, e.g code, test.
* Computer used.
* Word size.
* Unit size.
* Language used and SLOC.

The sizing process

1. **The input.** To obtain an estimate for a module the user indicates the selecting parameters, namely:

   * Index (identifier of standard function group)
   * Development status, indicating whether the project in the data-base is completed or not.
   * Language.
   * System name.

* Function name.
* Range of SLOC.
* Development computer.

2. **The process.** The system creates a temporary data file of the selected potential analogy entries, from which the user may chose a subset for statistical manipulation.
3. **The output.** Statistical manipulation on the selected data yields:

* Record count.
* Mean, median, variance and standard deviation.
* Beta distribution curve, using weighted averages .
* The most likely value for the size of the new function.

4. Graphic representation of the expected range of SLOC, based on standard deviation, at the confidence level prescribed by the user .

The Quantitative Software Management (QSM) Sizing Model

The Quantitative Software Methods (QSM) Size Planner [Put87] offers two separate methods for sizing products, which differ primarily with respect to the level at which they are applied, namely:

- Standard components are grouped at a detailed level of specification that allows comparison of attributes of a new project with the same attributes of completed projects. The compared components are: files; reports; screens; batch programs; modules; subsystems; SLOC; interactive programs; bits; bytes; words and object instructions.
- A predicative model to estimate LOC based on non-LOC based size at the application/functional level (e.g. Function Point Value).

Both methods are based on analogy, and use Fuzzy Logic theory.

The data-base

An historical data-base is used, comprised of eleven sub data-bases, each including one application category e.g. real time, business, micro-code. The statistics included in the data-base allow us to associate a range (size) to each of the categories. The size category and the size range are classified in Fuzzy Logic.

The sizing process

1. **The input.** The user defines:

   * The desired sub data-base according to the application category.
   * The overall size category (to be assigned to the application which is being sized), ranging from very small to very large.
   * A size range within the category and by this refines the overall size estimate.

   It should be realised that **judgement** is needed as early as the initialisation stage of the process.
2. **The output.** A mean and standard deviation is provided for the following:

   * Quantitative Software Managements (QSM) data-base statistics.
   * Estimates from user selection.
   * Combined weights estimate. This is calculated using the Bayesian formulae. A heaviest weight is given to the estimate which has the smallest standard deviation.

Discussion

The major component in all tools is the statistical data-base. The data-bases differ in their scope, the number of entries and the descriptors for each of the entries. One data-base is dedicated to the aerospace environment while the others include a variety of environments and applications. The descriptors of each entry and the way they are manipulated for the purpose of identifying an analogy depends on the approach adopted by its builders. All use statistical methods to describe the accuracy of the estimate provided. The analyst's (the user of the sizing model) judgement is essential in one or more of the steps in the process. However, the various models search for and identify the analogy in different ways. For example, the Quantitative Software Methods (QSM) model uses Fuzzy Logic, while Electronic System Division (ESD) and Software Sizing Model search for similarities to match a set of descriptors.

The input required for the ESD and the QSM tools to identify the project size at the system level is available relatively early in the life cycle. The data required is found to be easily understood [IIT87a]. It is also assumed that these two tools can be applied effectively without knowledge or experience in the application area, although the author of this thesis believes this to be somewhat dangerous.

4.2.3 Comparison of Project Attributes

The Comparison of Project Attributes approach requires a detailed historical data-base. The approaches within this category relate the current project to previous developments at either the system or component level. The approach compares projects in order to identify an analogy within project attribute categories. Statistical analysis is then used to yield an estimated size. The method differs from the tools described previously in details of the historic data-base used for Analogy. It affords comparison not only at the system or functional level but also at system components level and within the project attributes category. The

Computer Economics Inc. Sizer (CEIS) tool described below is an example of a tool in which this technique is applied.

The Computer Economics Inc. Sizer (CEIS)

The CEIS approach to estimate the size of a new project, is to compare attributes of a new task with those of three tasks existing in the data-base, whose their size is already known.[5] This implies a need for knowledge of the application area.

The sizing process

The sizing process comprises a comparison (and determination) of a size relationship between the attributes of the new task with three reference tasks of known size (completed tasks or to be developed). An Intensity of Importance (relative importance) value is assigned to the compared item with respect to: complexity; peak staff; technology rating; requirements volatility; specification level; required reliability.

The scale for the Intensity of Importance values ranges from one to nine, and its values are defined as follows:

1 = Equal Importance.
Two attributes or tasks contribute equally to the objective.
3 = Weak Importance.
Experience and judgement slightly favour one over the other.
5 = Strong Importance.
Experience and judgement strongly favour one over the other.
7 = Demonstrated Importance.

---

5. Lambert [Lam86] described the model at the 1986 ISPA Conference. It is based on the concept developed by Saaty [Saa80]. A technique incorporating a similar approach is suggested by Bozoki [Boz84;87] and is used by the Software Sizing Model (SSM) he developed.

Activity is strongly favoured and dominance is demonstrated in practice.

9 = Absolute Importance.
Evidence favouring one activity over another is overwhelming.

The values 2, 4, 6 and 8 lie in between the defined values and are used when a refinement is needed.

1. **Input.** The user is asked to compare a pair of attributes and to assign a numeric value using the Intensity of Importance definitions. This process goes on until all pairs have been compared and have been assigned a Intensity of Importance volume.
2 **Process and results.** The model compares, calculates and iterates between these steps to calculate the size of the unknown task. The process involves the following steps:

* Checking the matrix of the Intensity of Importance values assigned for consistency. If the consistency ratio is inappropriate, then the matrix is re-evaluated.
* Calculating the weights for each of the project attributes using Eigenvalue analysis. The Eigenvector is calculated and when normalised, so that the sum of the components of the vector equals one, the weights for each of the project attributes are found.
* Iterating, in a similar way, these steps to define the interrelationships between the three reference tasks for each of the attributes. When these relationships are determined and checked for consistency, the task to be sized is compared to each of the reference tasks for each of the attributes.
* The model calculates the size of the unknown task as function of the actual size of the three references tasks.

4.2.4. Size - In - Size - Out or Expert Judgement

The Size-in-size-out approach also initially asks for an approximation. The output is a refinement of the approximation given as an input. Statistics are used for the refinement process. A few techniques use this approach, e.g. Wide Band Delphi which combines several independent estimates from individual experts. Or, the Software Sizing Model (SSM), which involves ranking the software product or its components (e.g. modules) into a size order relative to some reference products as modules whose size is known. The Software Sizing Model now described is an example of this technique.

The Software Sizing Model (SSM)

The Software Sizing Model (SSM) was introduced in 1980 by Bozoki and revised in 1987 [Boz87]. The model offers a size estimate based on either SLOC or Function Point Analysis. The basic assumptions of this model differ from those described earlier. They are:

- The qualitative information available at the proposal stage is more accurate then corresponding quantitative data.
- An ordinal scaling technique is more appropriate for the size estimation process than a cardinal measurement. Small units that comprise the development process (e.g. modules) are often estimated more accurately in a **relative form** than those that relate to **absolute size.** This assumption dictates the point of time in the project SDLC at which the method can be used, that is, when the product can be partitioned into modules whose operational and functional characteristics are defined.

The sizing process

1. **The Input.** The users provide the model with:

* Module names and descriptions.
* Module size, if known. (Size is a mandatory input for at least two of the modules.

2. **The process.** The user is then asked to:

* Rank each module relative to the others. The modules are presented to the user two at a time, and he has to judge which is the larger of the two. This process goes on till all modules have been ranked from the largest to the smallest.
* Associate each module with a designated size interval.
* Provide a size range for each module, highest possible size, most likely and lowest possible.

3. **The output.** Based on the size of the modules already provided by the user as reference points, the SSM provides each module with the expected size and standard deviations and, for overall system the:

* Expected size.
* Standard deviation.
* Confidence limits, for each expected level of probability: low and high confidence limit.

The model uses statistical methods to map the relative size of each component to the reference modules of known size in order to obtain all module sizes, standard deviations and the total system size.

Although the input required is available early in the development life cycle and is easily understood, it is clear that the model cannot be applied without having a thorough understanding of the application area.

4.3 AN ALTERNATIVE UNIT OF MEASUREMENT FOR SOFTWARE PRODUCT

In the late 'seventies, a very different approach to project size was introduced. This was the 'Function Point Value' introduced by Albrecht [Alb79] and has been developed further since then [Alb83; 84].[6] Albrecht felt that the traditional LOC approach tended to penalise higher order languages and 'award' unusual code expansion due to use of code generators, re-use of code [Alb83]. A modified version of the Function Point Analysis was suggested by Symons in an attempt to overcome problems he identified [Sym88].

DeMarco [DeM82] takes a similar approach and suggested Bang delivered per unit of time as a productivity measure. The Bang is a quantitative indicator of the net usable function from the user's point of view. The Bang is based on 12 essential counts of primitives (p-count) that are adjusted by the number of input and output. It is a similar approach to that suggested by Albrecht, although the approach differs in the measure itself.

Reifer states in his wish list for cost estimating that *"better and more accurate ways of developing size estimates will be made available as research into Function Point theory begins to realise its potential"* [Rei87].

Emerging from implementing the prototype paradigm, a third approach to measure productivity might be offered: user specification per PM [Boe84].

4.3.1 Function Point Analysis (FPA)

The Function Points approach deals with **what** the system does rather than **how** it does it. The project size is measured by counting the functions and then is weighted for general complexity. However, it is worth emphasising that this

---

6. Two years after the Walston and Felix survey was published, another substantial productivity survey originated from the IBM environment. This dealt with projects completed over a similar time frame (74-79). Albrecht [Alb79], and Albrecht and Gaffney [Alb83] established the foundations of an approach to cost estimation called Function Point Analysis (FPA).

approach aims to **define a productivity** measure applicable throughout the project SDLC.

The model is linear using compensation factors for complexity. The relative size of the system (the product, the functions delivered) to be developed is determined by the product of two factors, the product **size** and the **technical complexity.** More accurately, FPA provides a more reliable measure than LOC of product size early in the SDLC. Indeed, this method has gained widespread acceptance as a way of measuring size, particularly in IBM-based MIS departments. The attractiveness of the Function Point as a productivity measure and thus as a means for estimating the software development effort, emerges from:

- Its basis on data existing in an organisation quite early in the project life cycle.
- Its simplicity of use. The assumption is that a non data processing user can evaluate the measure [Rub83].
- Its language independence.

Productivity is defined as the weighted sum of delivered Function Points units, per each of the classified information domains (inputs, outputs, internal files, external interfaces and enquiries, all divided by person-months).

(4.3) $Productivity = FP / PM$

(4.4) $FP = UFP * [\, 0.65 + 0.01 * SUM\ DI(f_i) \,]$

Where,

UFP = Total of all the weighted functions

f_i = the complexity adjustment value. The degree of influence associated with each of the 14 general characteristic variables which the authors [Alb83] believed affects complexity.

DI = The total degree of influence.

However, it should be noted that the Function Points are often converted to machine or assembly language LOC. The inverted LOC is then input into an effort estimation model to calculate the estimated effort.[7] Function Point Analysis is increasingly being used for sizing systems incorporating fourth generation languages [Ver87].

The Lines of Code and the Function Points approaches are distinct estimation techniques that both require the decomposition of software into modules or Function Points, that can be estimated separately. The two techniques differ in the level of detail required for decomposition. Estimates based upon lines of code can only be made, with a high degree of certainty, after the detailed design phase is completed. This means that only the estimation process for the effort needed during the Construction phase can be based upon LOC. The knowledge required for this method is available, with a certain degree of uncertainty, at the Project Planning phase. But it is only reliable after the Preliminary System Design is completed.

A weakness of the FPA method is that it does not consider environmental factors [Sym88].

Methodology

A heuristic approach was used to develop the model. The basic productivity standards were empirically obtained by analysing the history of completed projects, while the weights were determined by debate and trial. A prerequisite for applying the method is the existence of an historical productivity data-base of completed projects and a complete task schedule.

The Function Point is a 'surrogate size' measure from which the productivity is measured. Once the Function Point values have been calculated, they are used in a similar manner to LOC.

---

7. Albrecht used the average number of LOC required to develop a Function Point to show the relative productivity of COBOL, PL/1 and DMS/VS.

Assumptions

The project development process follows a 'disciplined management technique', a development process strategy following the Waterfall SDLC called: The Phase Approach. The SDLC is composed of phases, namely:

- Objective definition.
- System Design phase, which includes the following:

  * Requirements definition for customer approval.
  * External and Internal Design. The customer is provided with a proposed design of a system subject to his approval. This is done by providing the customer with a blueprint and proposal for further development and implementation.

- Implementation phase.
- Installation and maintenance phase.

The estimation process

The following steps are involved:

1. Assess the project risk.
2. Count the system components for each of the five classified information domains, namely:[8]

   * Number of user inputs.
   * Number of user outputs.
   * Number of user inquiries.

---

8. For full details of the Information Domains see **Appendix 4A.**

* Number of files.
* Number of external interfaces.

3. Compute the Function Point value by weighting each group of functions and summing the results, as illustrated in Table 4.1. This result is described as an 'Unadjusted Function Point', *(UFP)*. The weightings are subjective assessments based on the number of functions in each category.

		Weighting factors			
Function Domain	Count	Simple	Average	Complex	Total
User inputs	...	..x3	..x4	..x6	=....
User outputs	...	..x4	..x5	..x7	=....
User inquiries	...	..x3	..x4	..x6	=....
Internal logical files	...	..x7	..x10	..x15	=....
External interfaces	...	..x5	..x7	..x10	=....

Table 4.1 The Function Domain and their weighting factors.

4. Adjust the Function Point Value for complexity. This is achieved by estimating the 'degree of influence' of 14 'General Application Characteristics'. The 'total degrees of influence', *[SUM DI (f_i)]* is then converted to the 'Technical Complexity Factor' *(TCF)* using the following formulae.

$$(4.5) \qquad TCF = [\, 0.65 + 0.01 * SUM\ DI\ (f_i)\,]$$

The degree of influence is a linear scale from 0 to 5 and each degree of influence is worth 5% of a *TCF* which itself ranges from 0.65 to 1.35.

0 Indicates no influence
1 Incidental
2 Moderate

3 Average
4 Significant
5 Indicates essential influence

And,

(4.4a) $FP = UFP * TCF$

Productivity = FP / PM
Quality = $Errors / FP$
Cost = $\$ / FP$
Documentation = $Pages\ of\ Documentation / FP$

Discussion

Albrecht and Gaffney [Alb83] concluded that the Function Point approach enabled the determination of productivity trends for business applications in the IBM data processing services environment. This was the aim of the original research. Indeed, Function Point is used within IBM to measure efficiency (FP/PM), quality (errors/FP), productivity trends and maintenance. Pressman [Pre87] comments that as the method was designed and found suitable for business applications, it may not be suitable for applications of embedded systems. Albrecht's work was expanded to include embedded and real time systems [Rei87] with quite satisfactory results [IIT87a]. Symons [Sym88] comments that FPA is attractive as a productivity measure quite early in the project life cycle hence it is a means of estimating the software development effort. Furthermore, the overhead for data collection is low. However, Pressman [Pre87] takes a different view on this subject matter. He indicates that it can be difficult to collect the post-mortem data. Pressman points out that the opponents to the methodology find it 'subjective magic'. The same argument is presented by other researchers such as Verner [Ver89] and Symons [Sym88], who question the validity of the weights and their universalibility, i.e. their transferability to all environments and circumstances. The subjectivity of the weights assigned to the various Information

Domains is considered a problem. In addition, the following weaknesses of the methodology are noted:

- Over-simplification of the system components [Sym88]; [Ver89]. Symons replaces files by entity-relationships [Che77] in his FP tool (MARKII). Verner and Tate [Ver89] argue that such components should be further divided into primitive elements which differ for each function. For example: screens and reports should include the number of data elements, and files; reports should also include the number of data elements. In addition they take the view that the metrics should change with the technology.

  It is not at all clear that the 5 categories of function domains are sufficient to cover system functionality. The degree of function partitioning depends on the state of development.
- Redundancy of calculation, in determining the internal processing complexity. This factor is encountered while counting the UFP when weighting each of the domains and while adjusting for Technological Complexity. The use of a LOC count obtained from the conversion of FP to LOC in some the cost models can cause these system attributes to be reflected twice in the effort calculation.
- Overlapping between technological complexity factors.
- Adjustment scale for Degrees of Influence (DI). Symons argues that the range is not sensitive enough for the purpose it intended to serve. The weights aim to scale the system size and technical complexity. Some factors are included in the weights of the system size as well as in the DI, e.g. internal processing. The adjustment scale for DI is some times not adequate, as it does not provide enough range for difficulties existing between systems, related to the particular variable of f_i. For example, the effort needed to introduce a system to multi-sites will differ from the effort required to install a similar system in a single site.
- Underestimating the internal processing complexity factor. Symons [Sym88] suggests that this caused Albrecht [Alb84] to observe an unreal

decrease in productivity by factor of 3, when system size increased from 400 to 2,000 Function Points.

- The method takes little or no account of the development environment.

In general, however, the Function Point Value approach is widely accepted as a sizing and estimating tool. A study done by IIT Research Institute (IIT) [IIT87a] reported a fit of 28% and 30% between the actual and the estimated project size using ASSET-R (a tool for determining FP counts for real-time systems).

La Fourcade and Pickford of AT&T [LaF87] present an adaptation suitable to the Continuing Development strategy for the SDLC (often called Incremental Development). The strategy involves regular modification as a result of changing user needs. This adaptation addresses the need for additional function domains and the associated weights which will allow the counting of:

* Adding functions.
* Modifying functions.
* Deleting functions.

Each function domain will be counted and calculated separately but the total effort is the absolute value of the changing functions.

The Function Point Value is a way of sizing projects. A few of the sizing tools use modified versions of the FPA technique originally developed by [Alb79].[9]

4.3.2 Rubin's ESTIMACS Model

There are a number of widely used tools which are based on Function Point

---

9. For example: Quantitative Software Methods (QSM) and Caper Jones's SPQR SIZER - FP assess the complexity differently, while SPQR Feature Point and ASSET - R are modified versions adopted for real time applications. Before You Leap (BYL) developed by Gordon [Gor87], and ESTIMACS [Rub83] also use the Function Point Value technique.

Analysis. One example is ESTIMACS [Rub83;85] which is designed for use during Project planning (pre-requirements definition). ESTIMACS is a series of nine estimators: Small Project; Function Point Calculator; Financial Analysis; Maintenance; Portfolio; Risk; Hardware; Effort; Staffing and Cost Estimators.

The model validation is based on 5,000 development projects [IIT87a]. A major advantage of the model is its use at the outset of the project life cycle. However, its use for the development of embedded systems is questionable.

4.3.3 Converting the Function Point Value to SLOC

Software sizing tools provide a conversion table for Function Points to LOC in different languages, i.g. SPQR SIZER-FP, ASSET-R and BYL. Size prediction is based upon empirical observations of the relationship between various source languages and Function Points. The conversion table shows a difference in the estimated SLOC per Function Point as given by the various models and it illustrates the relative power of languages.[10]

The conversion supplied by the various models varies as the result of using different data-bases which may include subjective estimates for complexity and system characteristics, as well programs written by people of varying skills and language experience. Therefore, these conversion factors should be modified to fit the organisation which intends to make use of them.

4.3.4 DeMarco's Bang

DeMarco [DeM82] introduced a new measure of size that measures the net usable

---

10. For example, Basic Assembler is converted to 300, 320 or 400 SLOC. The last figure is the conversion for real-time applications. C is converted to 128 by two of the models but 90 by ASSET-R. The same phenomenon is shown for Pascal and MODULA 2, 91 by two of the models and only 70 SLOC by the ASSET-R model for Pascal while 80 and 65 for MODULA 2, FORTRAN and COBOL give almost the same SLOC.

function from the user's point of view, called the System Bang. The Bang metric is based on Structured System Analysis and the counts are derived from data flow diagrams, entity-relationships models and state diagrams. DeMarco's Bang is calculated from a specification or, if the system is strongly data oriented, from a count of objects in the data-base. Bang requires information which is not available early enough to be useful for sizing and estimating the cost for the feasibility study stage.

The system specifications are developed down to functional primitives. A Functional Primitive (FPR) is described as *"a trivial piece which is too small to justify further partitioning"* [DeM82]. The number of data tokens (input and output) associated with each Functional Primitive are summed and then used to adjust the value of the Functional Primitives. DeMarco provides a table of weighted FP increments for this adjustment and states that the values are based on Halstead's [Hal77] Volume/vocabulary formulae.[11] He defines *RE* as the number of inter-object relationships in the automated part of the data model and classifies the following:

- A 'Function Strong' developed product is determined as product having $RE/FPR < 0.7$.
- A 'Data Strong' developed product is determined as product having $RE/FPR > 1.5$.
- An 'Hybrid' product have *RE/FPR* in the range 0.7 and 1.5.

If a developed product is 'Function Strong', then the calculation of the Bang will be:

$$CTC_i = TC_i * \log_2(TC_i) \quad (4.6)$$

Where,

---

11. Halstead [Hal77] suggested a family of composite metrics to assess the complexity of the programming effort. The metrics consider both the data and the functional aspects of the programming. Halstead's work is discussed as a complexity metric in Paragraph 5.6.4.

TC_i = is Token Counts, input and output associated with the i^{th} functional primitive.

CTC_i = Corrected TC_i.

And,

(4.7) $Bang = SUM\ W_i * CTC_i$

The CTC_i is multiplied by its relevant complexity weighting W_i and the Bang is the sum of the weighted, corrected token counts. DeMarco defined 16 classes of primitive (see Table 4.2) although he suggests that the correction values for some classes may be less likely to remain invariant than others.

Class	Weight	Class	Weight
Separation	0.6	Synchronisation	1.5
Amalgamation	0.6	Output generation	1.0
Data direction	0.3	Display	1.8
Simple update	0.5	Tabular analysis	1.0
Storage management	1.0	Arithmetic	0.7
Edit	0.8	Computation	2.0
Verification	1.0	Initiation	1.0
Text manipulation	1.0	Device management	2.5

Table 4.2 Complexity weighting factors for various classes of functions

If the system is 'data-strong' (large data-base) then the Bang is computed on the count of objects in the **data-base** *(RE_i)*. Each object is adjusted for the number of relationships at the object boundary and the corrected objects *(OB)* are then summed. The *OB* is the number of relationships associated with each object in the automated **data-model**. The weighting factors for 'data-strong' systems are given in Table 4.3.

If the developed product is a hybrid it is advisable to calculate two sets of Bang metrics for the various characterised parts of the developed product. The two parts should not be combined and the development should be treated as if it were actually two projects.

RE_i	Corrected OB	RE_i	Corrected OB
1	1.0	4	5.8
2	2.3	5	7.8
3	4.0	6	9.8

Table 4.3 Weighting factors for 'data-strong' systems

Discussion

The Bang is environment dependent, therefore, the weightings should be calibrated to the particular organisation, including new weightings factors if necessary.

- DeMarco [DeM82] does not explicitly state the basis for the weighting figures. But, it appears that the weighting figures are generated by using RE_i+1 in Halstead's volume formulae.
- The Bang has some appeal as it is developed from specifications. But, the functional primitives are at such a level that the information would not be available early enough to be useful for feasibility sizing and costing.
- The approach would have gained more support if it was incorporated into an automated tool for structured systems analysis [Ver87].

DeMarco suggests that professionals should gain the skills to produce their own models that will be best suited to their own environments. He believes that even a relatively small project could invest in its own tools. The Bang is a general approach to sizing the software product rather than a precisely defined procedure. The user can customise the approach to his own needs and environment in various ways.

In the long-term, this could provide a system that is indeed well suited to each project, but it has disadvantages:

- It takes time (and hence cost) to set up such a system and to thoroughly test it.
- There are no initial default values until reliable data about projects developed in the specific environment is available.
- The person setting-up the model needs expertise.
- A large organisation will end up with a wide variety of different models, with little or no compatibility with other systems.
- The tools will normally be slower and less easy to use, with fewer spin-off benefits, than professionally produced tools.

4.4 CONCLUSIONS

A consensus exists about project size being the one factor that obscures all others affecting the software development effort. But, do we know what the project size will be when we start the development process? Do we have the ability to estimate it? A unanimity exists here that it is very difficult to estimate the project size. It is particularly true for product size measured in LOC at the outset of project life cycle. Estimating the number and types of functions associated with a software development is not easy either. The product size, which is the major input to the effort estimation formulae, is in itself an estimate. Even as the development process matures, the process of sizing software is subject to wide margins of error. Many factors contribute to this phenomenon but the unifying characteristic of all of them is the potential for error inherent in any subjective human reasoning process.

In addition, the difficulties associated with standardisation in counting of the LOC (see Section 4.1) and the nature of software as a human problem solving activity, makes it difficult both to identify useful measurements and to produce accurate estimates. These are the prime causes for poor input estimates to the effort and cost model, which is a real weakness of all the models. The implication from this is that we may need to accept a substantial degree of imprecision in the

best estimates for effort required for building software.

Life cycle phase for implementing. The sizing tools, that provide estimation facilities using the analogy concept, can be implemented during the 'requirements analysis' stage of system development. Examples are the ESD, SSA, SSM, QSM (Fuzzy Logic) and CEIS models, although they differ in the analogy technique used. Tools which base the sizing estimates on Function Point Analysis techniques can be applied, when the 'external design' is complete. ASSET-R uses counts of the number of operators/operands. QSM (Standard Components) is based on a count of components. Therefore, both tools can be implemented only when the detailed design is complete.

The unit of measure for software product. LOC is still a standard measure of product size: it can be measured for the programming phase of the software development effort. The trend towards the replacement of the LOC as the unit of measure for the software product is discernible, at least among software organisations, where the majority of the applications are business oriented. The spread of the function based estimation models is evident. LOC and Function point are useful measures, however, for estimating the effort in early stages of the SDLC non LOC size metrics are preferable.

Chapter 5

CRITIQUE OF PARAMETRIC MODELS AND COMPLEXITY

5.1 INTRODUCTION

After our consideration of the development of effort estimation modelling, models for assessing productivity and methods for sizing software products, it is now appropriate to assess current estimation practice. Three major issues are the concern of this chapter:

- The problems with the current practice are summarised. The results of comparative studies and of the major findings from models are presented and analysed. Conclusions are derived for the transportability of cost estimation models, the relative efficiency of the models and the need for calibration.
- Resource allocation among phases of software development is of particular interest for this research and is analysed in detail.
- Several themes concerning effort estimating have emerged from the previous discussion (in Chapters 2, 3 and 4). These are aspects of the uncertainty, feedback, entropy and complexity associated with software development and their affiliated measurements. Alternative approaches to complexity are presented and their limitation for the purpose of this

development and their affiliated measurements. Alternative approaches to complexity are presented and their limitation for the purpose of this research is emphasised and criticised. The complexity determinants are examined and their implication for usage in estimating the effort required for the development of a project early in the life cycle is addressed.

5.1.1 Problems with effort estimating - the current practice

Managers at all levels in the organisation and others who are associated with the software development process, and hence with estimating the effort required for this process, are in need of support in this area. Good estimates of both the cost and duration of the development effort are considered critical to the success of software development. The existence of so many models shows that companies involved in large scale software projects require some mechanism to predict effectively, to manage and to track the amount of time and cost associated with software development. Software effort and cost models may support the management of software development throughout the process. However, the level of confidence in the estimates differs, based upon the stage in the life cycle at which the estimates are made, the amount of reliable information known to the estimator at that time, the dynamics of the environment and the inherent **absolute and relative uncertainties** [Leh89] associated with the process of building software.

Both the practices of sizing the target project and of estimating the effort needed involve tradeoffs between system specifications, aspects of software quality and scheduling within resource constraints. This implies that **human interactions and judgement** are essential throughout the estimation process. Interactions are a source of **entropy** in the process of software development. Judgement implies feedback and recycling to previous stages, and so, is also concerned with entropy.[6]

---

6. Webster defines entropy as a measure of the disorder of a closed thermodynamic system [Web79]. It is related to such things as: friction and resistance, heat loss, turbulence and random motion and disorder. The concept of entropy is borrowed and is used as a concept in management of software development as is further discussed later in this chapter (Paragraph 5.5.2) [Ton79]; [Bel79a]; [Bro75]; [Moh79].

as a reason for poor estimates of the effort needed for software development. People involved in this process come from various professional backgrounds. Their interest in the development stems from different objectives and from various political sources. The various parties associated with the software development speak 'different languages'. They may have different objectives in estimating the effort and may interpret issues differently. Often, the meaning of an estimate is not clear. Do estimates represent the lowest effort and the shortest duration for the development process? Alternatively, do they portray the scenario for the worst case? Problems stem from an attempt to impose a target rather than to estimate a range of effort values representing the estimator's belief of what is likely to happen. Analytical issues stem from the use of different terminology causing difficulty in identifying the activities associated with an estimation model, or interpreting the model's parameters. The Tower of Babel syndrome is a possible outcome of the process of estimating the effort for software development. Problems which are often encountered include underestimating the effort and the inability to improve the estimates throughout the development process. Estimating and maintaining software effort consumes the time of experienced people. Managers often do not want or are not able to approve this costly procedure, which might be 5% to 10% of project development costs [Kit89].

Independent empirical studies indicate that the models yield substantially different estimates.[2] Some reasons for this phenomenon are:

- Difficulties in applying consistent rules for estimating and counting the LOC and the effort required to develop them.
- Inconsistencies in using subjective adjustment factors.
- Differences in environments. The environment in which a model was developed and the environment in which the model is to be used often differ. This may mean that the adjustment factors are not appropriate for use in the different environment.

---

2. [Kit85]; [Miy85]; [Rub85]; [Con86]; [Kem87]; [Fun87].

Accepting these reasons implies that a practical approach to minimising the inaccuracy can be found.

The current practice does not emphasise the early stages of project development. The prime reason for this is understandable. The lack of reliable information about the problem to be solved, the environments it intends to serve, the alternative solutions for technical design issues, all emerge in uncertainty. Hence, estimating the effort required at the outset of software life cycle is a very difficult task with a minimal perceived chance for success.

5.2 EVALUATION OF MODELS - EMPIRICAL STUDIES

Evaluating the current estimation tools will take us one step further towards understanding the factors affecting the software development process. The studies are of three overlapping types:

- Studies which compare estimates using hypothetical problem statements. Rubin's [Rub85] study belongs to this category.
- Studies which use actual data from development projects to validate models and tools, or to calibrate them to a particular environment, e.g. [Kit84;85]; [Miy85]; [Con86]; [Kem87].
- Studies which compare estimates from two or more estimation models and or tools, such as the studies of Kitchenham [Kit84;85]; Conte [Con86] and Kemerer [Kem87].

Other studies which discuss specific languages (e.g. ADA) or maintenance effort are excluded and are only mentioned to indicate interesting related findings, e.g. Mohanty's [Moh81] and Funch's [Fun87] studies.

5.2.1 Rubin's study

Rubin's study [Rub85] is based upon a hypothetical problem. The objective was to present a general comparison of models. The paper served as the basis for a panel discussion at the 8th International Conference on Software Engineering. It was not intended to judge accuracy. The statement of the problem contains information at two points in SDLC, the 'conception' and the 'post external design', and includes quantitative data as well as qualitative information relating to the constraints and the environments.

All the compared tools, with the exception of ESTIMACS, are based on LOC as the unit of measure for project size. The SPQR tool has the ability to estimating size in either Function Points, or LOC. Yet, these two tools (ESTIMACS and SPQR) differ in the method used to count the Function Points and hence, to estimate the product size. ESTIMACS uses a high level business description and the answers to a set of questions as the basis for the Function Point counts and their complexity adjustments. SPQR requires the Function Point type counts as an input, although it is not differentiated by complexity. The study results in the following:

Tools	Effort (PM)	Duration (Months)	Peak Staff
JS-2	940 *	31	43
SLIM	200 *	17	17
GECOMO	363	23	22
ESTIMACS	112 **/***	16	15
PCOC	345	23	24
SPQR/10	437	28	16

* Minimum time solution
** Application structure was assumed to have average complexity.
*** For convenience the original figure (in hours) was converted to months using COCOMO's average of 152 hours per month.

Table 5.1 Rubin's Estimation Results

These tell us more about differences in the environments from which the tools were obtained and calibrated than it does about any particular tool. It should be noted that the study could not judge accuracy, given that the projects did not take place.

5.2.2 Kitchenham's and Taylor's Study

Kitchenham and Taylor [Kit84;85] advocate the need for calibration. Their conclusion stems from a comparison of the COCOMO and SLIM models implemented in British Telecom and ICL environments. The two studies covered 34 projects, with considerable differences in the data. The ratio of SLOC per PM varied by a factor of over 30 for the BT data and by over 20 for the ICL data. Kitchenham and Taylor show that for both models the estimated cost and schedule worked out much higher than the actual effort in almost every case. Hence, calibration was required in order to make sensible predictions, but needed, a fairly large historical data-base. There appears to be significant differences between the two environments with ICL being significantly more productive than BT, though not necessarily on the same types of applications.

5.2.3 Miyazaki's and Mori's Study

Miyazaki and Mori [Miy85] made an extensive evaluation of Intermediate COCOMO using data from 33 projects which served as the basis for the COCOMO model. Using the non-calibrated model they observed an average deviation of 166% between the estimates and the actual data, and in only 6% of the cases was it less than 20%, which emphasises the need for calibration as an adaptation process. These figures differ from those achieved by Kemerer's study [Kem87].[3] As a result they calibrated the COCOMO model to their environment

---

3. See Paragraph 5.2.5.

by:

- Eliminating a number of cost drivers, including: virtual machine volatility, analyst capability and main storage constraint.
- Changing the influence values of the various cost drivers.

The calibrated equation for nominal effort, is:

$$(5.1) \qquad Effort = 2.15\ S^{0.94}$$

Where,

$$S = KDSI^{*}$$

* Miyazaki and Mori support Boehm's practice of counting only one third of the lines of code in the COBOL declarative divisions.

Note that the exponent value in equation *5.1* is less than one, which indicates 'economies of scale' for their environment, in contrast to the 'diseconomies of scale' shown in data sets for the different environment for which Boehm originally calibrated the Intermediate COCOMO ($Effort = 3.2\ S^{1.05}$). By calibrating the results, an average deviation of only 17% is found. This means that where data from the history of projects exists in an organisation, a calibration process could improve the results of the model's estimates to an acceptable level.

5.2.4 Conte et al.'s study

Conte et al. [Con86] calibrated and applied both SLIM and Jensen's models for six separate data-sets, from widely differing environments, 187 projects in all. The evaluation of tools was based upon a set of evaluation criteria suggested by Conte. The results are given in Table 5.2. Conte's evaluation criteria are:

* the Magnitude of Relative Error *(MRE)*.
* the Mean Magnitude of Relative Error *(MMRE)*.
* the Prediction level of a model *PRED(level)*.

(5.2) $MRE_i = |E_i - E'_i| / E_i$

Where for project i,

MRE_i = the magnitude of the relative error.
E_i = the actual value of a cost per parameter.
E'_i = the estimates of the cost parameter.

(5.3) $MMRE = 1/n \sum_{i=1}^{n} MRE_i$

Where,

n = number of projects in the data-set.

(5.4) $PRED\ (level) = k / n$

Where,

k = the number of projects in a set of n projects whose $MRE_i <= PRED\ (level)$

The prediction at *PRED (level)* means that if *PRED (0.25) = 0.85,* then 85% of the predicted values fall within 25% of their actual values.

Conte et al. suggest that *MRE <= 25* and that *PRED(0.25) >= 0.75* are acceptable for an effort prediction model. This criterion would permit some extremely poor values since, there is no limit on the *MRE* of the other estimates that exceed 25% of the actual values.

Both tools exhibit poor predictions of effort for all the data sets. The SLIM effort prediction had a mean error of around 0.9 of the actual value and a *PRED(0.25)* of 0.10, which means that only 10% of these estimates are within 25% of the actual effort expended on the project. However, Jensen's results are slightly better than that of SLIM, with a mean error of around 0.8 and a *PRED(0.25)* of 0.17. Conte et al. concluded that SLIM over estimates effort for small and medium size systems, exaggerates the effect of schedule compression on effort and is quite sensitive to the choice of level of the Technology Factor.

This comparison emphasises that models for estimating are not easily transferred to different environments and that their calibration is essential.

Data-base	No. of Projects	Jensen MMRE	Jensen PRED(0.25)	SLIM MMRE	SLIM PRED(0.25)
Boehm (COCOMO)	63	1.01	0.10	1.04	0.06
Belady - Lehman	33	0.76	0.06	0.88	0.06
Industry (Anon)	40	0.76	0.10	0.83	0.05
NASA/Goddard	19	0.63	0.21	0.78	0.05
Yourdon 78-80 survey	17	0.70	0.24	0.78	0.24
USA Army	15	0.80	0.33	0.97	0.13

Table 5.2 Conte's calibration of Jensen's Model and SLIM

5.2.5. Kemerer's Study

Kemerer's [Kem87] study aimed to evaluate the accuracy of the model's estimates of effort outside their original environments. The study evaluated four estimating approaches: The SLIM and the ESTIMACS tools, a tool based on COCOMO and the FPA method. This selection represents a symmetry from two view-points, the unit measure of project size and the proprietary nature of the model. Two of the tools, the COCOMO based tool and SLIM, use LOC as the unit of measure for the project size, while the other two use non LOC units of measure for the same purpose. The COCOMO based tool and the FP method are nonproprietary while the other two are proprietary. Fifteen projects were chosen, 12 of which were data processing projects written entirely in COBOL. The average project size was under 200 KSLOC. The project data originated in two companies: a national computer consulting firm and a consultancy firm ABC. The tests conducted were:

* *Percentage error* = $(PM_{est} - PM_{act}) / PM_{act} * 100$
* MRE^* = $|(PM_{est} - PM_{act}) / PM_{act}|$
* *F-test*

*Errors in estimating can be of two types: under-estimates and over-estimates. The two types of errors do not cancel each other out when an average multiple error is taken.

The study results

The study results show the following percentage errors:

* SLIM 772%
* COCOMO 603% as average for the three COCOMO models with the lowest error being 83%.
* FPA 167%
* ESTIMACS 85%

None of the models give good effort estimates. All overestimated substantially, from 85%-772%. SLIM over-estimated all fifteen projects. This can be explained by the data-base used for the development of SLIM (Defence related projects) in which productivity is usually assumed to be lower than in other environments[4]. COCOMO (both the Basic and the Intermediate models) provided similar results presumably for similar reasons to the SLIM model. Effort was overestimated in all 45 cases. The Function Point Analysis method produced substantially better results than either of the two models which use SLOC.[5] This is probably due to the fact that the Function Point Analysis was developed in a similar environment to that of the study test. However, between the two PFs based models, the

---

4. The SLIM data-base currently contains projects from all sectors of industry which is used to calibrate the tool to a particular environment.

5. The FP method produces size, not effort. The effort (PM) is generated by performing a linear regression with the PM as the dependent variable and FPs as the independent variable.

ESTIMACS produced better results, probably due to the estimators' experience with the application area (insurance).

The results after calibration of the models proved to be significantly better. Paradoxically, SLIM, which gave the worst result before calibration, correlated well with the actual effort after calibration, explaining 88% of the behaviour of the actual person month effort in the ABC data-base. The SLIM estimates were used as the independent variable and the actual PM as the dependent variables using a regression equation.

The *F-test* checked the null hypothesis that the ABC and the Function Point Value models are the same. The two individual regressions (one of the Function Point Value data and the second of the ABC data), were compared with a single regression composed of all 39 data-points. The null hypothesis could not be rejected at 95% confidence level.

ESTIMACS evaluation (including only nine projects) produced the better results. This may again be due to the similarity between the ESTIMACS data-base and that of ABC.

Effect of Complexity factors

The various factors for complexity adjustment proved to have no effect on the estimates produced in the study environment by the various models. Also noteworthy are the results of an additional test run by Kemerer which aims to validate Albrecht's assumption that the Function Points count is highly correlated with SLOC. The unmodified Function Counts were found to have higher correlation with the actual effort than the modified Function Points. Similarly, the cost drivers in COCOMO appeared to have little effect on the estimates. The Intermediate and the Detailed COCOMO models were not significantly better than the Basic Model, and in fact correlated less well with actual PM. The Function Point method provides better estimates than LOC, although ESTIMACS uses 20 additional questions which aim to assess productivity. From this Kemerer concluded that Albrecht's complexity adjustment factor and COCOMO cost

drivers do not add any information in this particular case. Hence, they do not model the factors affecting productivity very well.

5.3 TRANSPORTABILITY OF COST ESTIMATION MODELS

The inability to transport cost estimation models is well supported by the independent empirical studies discussed previously, which in addition, pointed out the incompatible nature of the models. In nearly all instances, the models were formulated in the late 1970's for a particular environment and the software development methods used at that time. For example, although the prototyping approach was known, it was not widely used. Most of the models do not specifically account for prototyping and/or incremental development. But, the accumulated effort required when following these strategies differs from that observed when using the traditional development strategy.

5.3.1 The relative efficiency of the models

The results presented in Kemerer's study indicate that the two models which based their estimates on size measured in Function Points resulted in estimates with lower error before calibration, in terms of MRE. In terms of the regression results, both LOC based models correlate higher than either of the non LOC based models. Yet, it is important to recognise that the Function Points counts can be obtained early in the project's life cycle, while the LOC estimates can only be obtained later in the development process. This implies that the models which are based on the Function Points method represent the better alternative to size estimates early in the SDLC.

Some of Rubin's [Rub85] results conflict with those indicated by Kemerer [Kem87]. Rubin points out that the SLIM model forecasts a lower effort than the two COCOMO based models. SLIM's prediction is 200 months with minimum time constraints, while GECOMO and PCOC predict 363 months and 345 months

respectively, without considering time constraints (see Table 5.1).[6] Further analysis of Rubin's results, shows that the two tools in which a minimum time solution is suggested, yield very different results. The JS-2 tool suggests the minimum time solution to be 940 months, while SLIM suggests it to be only 200 months. Is this a property of the model's basic formulae for effort prediction? Or is this a result of the model's different perception of the relationships between effort required for the development and its duration? It could be assumed that the compensation factor in the JS-2 tool for shortening the elapsed time is much higher than that assumed by the SLIM formulae, but, this assumption contradicts our previous analysis. The SLIM's effort/time relationship is highly penalised for shortening the project's duration. The constant used in the JS-2 tool *(3.27; 3.28)* is less extreme than the formulae *(3.25)* used in the SLIM tool.

5.3.2 The need for calibration

The second major conclusion of these findings is the need for calibration when a model is used in different environments. Indeed, a consensus exists about the need for calibration. Developers of the estimation models recognised the need to calibrate their models to a new environment [Put78]; [Boe81]; [Frei79]. The calibration is done by assigning values derived from projects implemented in the environment in which the model is to be used. The need was underlined by a number of studies, Rubin [Rub85], Kemerer [Kem87], Kitchenham and Taylor [Kit85]. Miyazaki and Mori [Miy85] report a meaningful improvement in estimates resulting from calibration. Funch [Fun87], Acosta and Gloub [Aco87], Desharnais [Des88], Jeffery and Low [Jef89] also concluded the need for calibration. Funch [Fun87], who studied 26 US Air Force and Mitre Corporation projects, reports that calibration substantially improved the results. Yet, he concluded that coefficient calibration was superior only to coefficient and

---

6. Kemerer's study results before calibration are: SLIM > COCOMO > FPA > ESTIMACS.
Rubin's study results are: COCOMO (based models GECOMO and PCOC) > SLIM.

exponent calibration. The coefficients Funch found to be suitable for effort estimation are shown in Table 5.3.

	Effort Basic mode		Effort Intermediate mode		Schedule Intermediate mode	
	Funch	Boehm	Funch	Boehm	Funch	Boehm
Embedded model	6.5	3.6	2.4	2.8	3.8	2.5
Semidetached	2.4	3.0	3.1	3.0		

Table 5.3 Funch's nominal effort and schedule coefficients compared with Boehm's originals coefficients [Fun87]

An investigation of the factors influencing the software development effort has shown the difficulty in isolating, determining and measuring the effect of **a particular factor** on software development effort, as these factors are inter-dependent. Models developed in a certain environment will only be able to reflect the impact of the factors which have variable effects within that environment. Factors which are constant and hence, do not cause variations in productivity among projects produced in that environment, may have variable effects in another environment. In addition, each organisation has its own standards, procedures and culture, thus, scaling and calibrating would be necessary. This may clarify why different organisations find some models more effective than others.

It is possible to adapt a model that has been developed elsewhere in order to improve the process of estimating. The results after calibration (such as the SLIM improvement in the studies of Kemerer [Kem87] or Miyazaki and Mori [Miy85]), probably justify considering parametric models, if there is sufficient historical data for the calibration process. Yet, the cost of those improvements should be taken into account. A model for effort estimation should only be adapted if the cost of adaptation is less than that of developing a new model from scratch.

5.4 RESOURCE ALLOCATION AMONG PHASES

The interest of this research lies in understanding the resource allocation between the Preliminary System Design and the rest of the SDLC. Each of the tools discussed uses an empirical approach which tries to identify the activities associated with the process of building software. Historical data is then used to determine the percentage of the effort which was expanded on each activity or, more specifically, on each set of activities. The various tools use different averages of resource distributions over the SDLC, based on the environment from which they were obtained. What can be inferred from these distributions for the purposes of this research? It will not be easy, since analytical issues arise. Terms used in the context of effort estimation are often ill-defined. Although these terms may have a precise meaning in some models, they normally differ from model to model. However, it is worth trying.

Tausworthe [Tau83] offers the following as a typical top-level Work Break Structure skeleton (WBS), as shown in Table 5.4. The first three tasks comprise what is considered in this research as the Project Planning (PP) and the Preliminary System Design (PSD). The typical accumulated percentage for these two phases is 26.5%.

	Percentage			
	Effort	Time	Effort **	Time
	1	2	3	4
System planning and requirements	8.5	9.5	9.5	10.5
Software planning and requirements	5.4	6.7	6.0	7.4
Software architecture and design	9.9	9.9	11.0	11.0
Detailed software design and production	47.0	29.8	52.2	33.1
Software test and transfer	19.2	34.1	21.3	38.0
Management tasks and milestones	10.0	10.0	--	--

* Time will vary if task precedences are changed so tasks become more sequential or more concurrent.

** Columns 3 and 4 represent the resource allocation excluding the management task, as this task is not included in any resource allocation presented hereafter.

Table 5.4 Typical top level breakdown structure, after [Tau83]

Wolverton [Wol74] presents a typical allocation of resources in customised software development as shown in Table 5.5. His way of phrasing activities differs from Tausworthe's, and causes analytical problems. The first two or three tasks contain what is considered in this research as the PP and the PSD. The range of the accumulated typical percentage for these tasks in is 26.0 - 30.0%.

	Percentage of effort (total)
Requirements analysis	8.0
Preliminary design	18.0
Interface definition	4.0
Detailed design	16.0
Code and Debug	20.0
Development testing	21.0
Validation testing and operational demonstration	13.0

Table 5.5 Typical resource allocation for customised development [Wol74]

Walston and Felix [Wal77] suggested resource allocation, as shown in Table 5.6. The first two tasks contain what is considered in this research as the Project Planning and the PSD. Their accumulated equivalent percentage is 26.0%.

	Percentage of effort	
Requirements analysis	7.9	10.1 *
Preliminary design	12.1	16.0
Detailed design	12.9	12.9
Code	13.1	13.1
Unit test	24.3	24.3
Integration	13.0	13.0
Test	11.6	11.6
User Documentation	6.0	--

* The right column represents the resource allocation where the 'user documentation' task is included in the two first tasks. 'User documentation' is part of the 'requirements analysis' and the 'preliminary design'.

Table 5.6 Typical resource allocation, after [Wal77]

Summing the effort expanded prior to the 'detail design' resulted in:

	Percentage	
Tausworthe	26.5	(including 8.5% for planning)
Wolverton	26 - 30.0	(including 8% for planning)
Walston and Felix	26.0	(including 7.9% for planning)
Boehm, Organic Mode[7]	22.0	(including 6% for planning)
Boehm, Semidetached Mode	24.0	(including 7% for planning)
Boehm, Embedded Mode	26.0	(including 8% for planning)

Table 5.7 Comparison of typical resource allocation prior to the 'detail design'

It is not always clear what activities are included in an effort estimate provided by a model. Project Planning ('concept development') and 'requirements specification' are not covered by any of the prominent models, yet the proper specification and documentation of these requirements are essential to the proper development of the software product. Ambiguity exists also for the 'direct' software development activities. It is not clear which activities are included in the effort estimate and which are not. For example, activities often excluded from the estimates are those associated with the tendering process, the cost benefit analysis or the independent Verification and Validation set of activities which typically consume 25% of all project resources [Fai85].

5.5 UNDERSTANDING COMPLEXITY

The particular part of the world we are going to address is that concerned with the management of human-activity systems which we try to create and control in order to achieve some collective purpose. The management of such systems is constrained by the limits of our perceptions and understanding of actions and reactions, and by the beliefs we have about causes and effects. ... The problem of dealing with the world of complexity and surprises: a world for which we

---

7. See Table 3.6

> *have no ready-made models and pat formulae, and for which there is as yet no great body of received wisdom to which to turn* [Rob88].

Understanding complexity is a worthy exercise in its own right. Yet, the interest of this research lies in understanding complexity and how it affects the process of software development. It aims to find points of view which will be of help in choosing an approach for interpreting complexity in the proposed model for Effort Estimation (EEM). It is perhaps most appropriate to start this discussion with a series of questions: What is meant by complexity in the context of software development? Are the effects of the determinants of complexity on the process of software development static, or are they dynamic? Are we able to identify the causes of complexity? And if so, can we isolate and classify the effects of each of the factors determining complexity and then measure them?

Complexity, as introduced in Chapter 1 (1.8), is generally used to point at foreseen difficulties in the development process caused by a variety of interacting agents affiliated with software development. In other words, process complexity equals risk. Understanding and managing complexity is considered a key to large software endeavours [McG80]. The ability to estimate the effort required for software development is highly dependent on the ability to identify areas of potential complexity and to prepare for them as the:

- Productivity rate of software development is a function of the system complexity (system difficulty) and of the uncertainty associated with the development process and as such has an impact on the effort and the duration of software development.
- Complexity in system development may cause a reliability problem.

5.5.1 Uncertainty

This research concentrates on estimating the effort required for the Preliminary System Design phase, at the Project Planning phase. Uncertainty is inherent in the

process of software development. However, the level of uncertainty associated with the PSD is much higher than at the later phases. There are many reasons for this. The problem is often unstructured and poorly defined. It is in this phase where solutions to the problem are identified and negotiated. Intensive **learning processes** take place as these are the least understood facets in the development process. The users are learning about the potential of information systems, the alternative solutions, the resources required for the various alternatives and their implications on the application area. While this is going on, the software engineers and the system designers are learning about the application problems, to which a solution is needed. These learning processes are unavoidable and so is the uncertainty.

The inability to specify these needs in formal language is an additional cause for the high level of the **uncertainty** which resides early in the SDLC. At the start of software development, the organisational objectives are transformed to user needs and requirements, which are then transformed to system specifications, and which can be expressed only in verbal form, in natural language. This situation is theoretically unique to the the first stage of system development, as system specifications can be expressed in a formal method which avoids ambiguity in the transformation process to the next stage [Leh89]. But, although supporting tools and formal methods are available, they are not widely used. **Ambiguity** is, therefore, difficult to avoid and in practice, **feedback** is needed throughout the development process, for example: the Verification and Validation (V&V) processes. Hence **entropy** is generated which results in inherent **uncertainty.**

5.5.2 Feedback and entropy

> *Systems with feedback, driven far from equilibrium, may become unstable and undergo spontaneous transitions to produce new, more complex processes which are also stable. They exhibit the property of self organisation. How this arises in the natural world is the subject of heated debate at present. .... It seems clear that the complex system may suddenly possess the property of spontaneously generating systems of a higher order and that the emergence of hierarchical organisation is a far-from-rare event at all levels in the physical*

> *and biological world and, if we accept the power of analogical reasoning, in the organisational world as well* [Rob89].

The previous discussion and analysis leads to the understanding that each process of developing a software system models the complex environment it aims to serve and includes an implicit model of itself, with built-in feedback mechanisms. Changes occur during the process of software development (and operation), which result in changes to previous decisions and actions, and to objects that have resulted from these actions. These changes result from the very nature of computerised systems being used to address problems in working environments in which human beings are involved and which are never static.[8] It is noteworthy that the changes are, neither solely nor mainly, caused by previous mistakes or imperative understanding of the problem.

Every interactive process results in some amount of **entropy**. In the social reality, where the process of software building occurs, uncooperativeness, incoherence, confusion, indirect or disordered action are some of the issues which will cause entropy. Interactions and communication among the parties associated with the development process such as managers, project teams, users and secondary contractors, are required to negotiate a solution for the stated problem which the target system aims to solve. For additional problems not addressed in the problem statement but raised throughout the development process, they may yet require solutions.

The communication and the interaction among the parties involved in the two processes, the development process and the effort estimation process, are additional causes of loss of resources. Each of the processes represents different resources which interact, aiming at building a software product, thus, yielding entropy. The software development process represents the manpower resource, and the effort estimation process represents the other resources which are associated, such as budget and **manpower available** for the development and duration of a project. The interaction between the two processes yields a cascade

---

8. See the definition of a system in Section 1.2.

of changes. A new cycle of negotiation, for either additional resources or a design-to-cost process, may start when the effort required for the negotiated and agreed upon solution is estimated, although, the allocated resources are insufficient, or when there are time constraints on the development process. Hence, it is certain that **feedback** and **entropy** will be among the products of these activities and, thus, **uncertainty.**

Tonies [Ton79] suggests that the goal of software management is to minimise the 'total integral of entropy in the system'. He identifies groups of factors which induce confusion and disorder into the management of the software development process. These are common causes of entropy. The factors considered are whether: the software task is under-sized and the requirements baselines are poorly analysed and therefore, the project resources do not match the software task; the methods used for design and programming are inappropriate; the test disciplines are poor; the communication between the parties is poor; the configuration control and the project management controls are ineffective. The characteristic causes for these factors are proposed. For example, some of the characteristic causes suggested for under sizing the software task are short estimates for project schedules and insufficient budget for the development of the product. The characteristic causes for ineffective project management control are such as: poor visibility of product status and uneven support of management by staff.

The notion of entropy can be used in various ways. It can provide a measure of the amount of uncertainty within a system and a probabilistic measure of complexity [Bel77;79a]. Belady used entropy to model the uncertainty involved in the decisions taken in the various stages of introducing a change into a system.

A different approach is presented by Mohanty [Moh79] based on the work of Channon [Cha74] and Schutt [Sch77]. The entropy within a system is defined in relation to the number and the importance of assumptions one subsystem must make concerning another.[9] This measure can be used to compare the quality of different arrangements for the structure of the system. However, it is difficult to

---

9. See definition of complexity in Section 1.8 for similarity between these approaches.

identify and assign proper importance weights to assumptions.

5.6. ALTERNATIVE APPROACHES TO COMPLEXITY

Researchers have tried to identify complexity factors, to propose ways of measuring them and to relate these measurements to the actual development effort and quality. Most of the studies focus on measuring complexity of the programming process, only a few have concentrated on measuring complexity of developing specifications. The various approaches suggested to measure the complexity associated with the software development are the subject of the following paragraphs.

5.6.1 Logical complexity

Farr and Zagorsky define logical complexity as: *"A measure of the degree of decision-making logic within a system"* [Far65]. This definition refers to the decisions count of a program. The density of IF statements (logical branches) is the measure of the logical complexity. Farr and Zagorsky found it to be a significant predictor for program cost.

Although the metric of IF statements has the characteristics of being easily countable through computerised algorithms, the density of IF statements is impossible to estimate at the start of the project life cycle. This metric does not represent the difficulty for the PSD effort. Therefore, the metric could be of value as a post productivity measure only.

5.6.2 Structural complexity

Gilb defined structural complexity as *"A measure of the degree of simplicity of relationships between subsystems"* [Gil77]. The term Relative Structural Complexity

(RSC), is defined as:

(5.5) $RSC = ML / M$

Where,

ML = Number of modules linkages

M = Number of modules

The **absolute structural complexity** is defined by Gilb as the number of modules or subsystems. Its aim is to indicate system maintainability, flexibility and adaptability. This definition introduces complexity as relationships between subsystems, which is foreseen as an important element. Gilb does not support his definition with any empirical or theoretical findings.

5.6.3 Cyclomatic Complexity Value

A different approach to the logical complexity is offered by McCabe [McC76] and has been further developed by Chen [Che78]. McCabe defined complexity in relation to the decision structure of a program. Based on Graph Theory, the metric counts the number of distinct control paths in a module. The number of regions computed from the planar graph is the Cyclomatic Complexity level *V(G)*. [10]McCabe's argument is that the higher the decision count, the more difficult it is to test and to build a reliable module. Cyclomatic Complexity may be computed by one of the following algorithms:

(5.6) $V(G) = E - N + 2$

Where,

$V(G)$ = the number of 'regions' in the planner graph of the control flow G. 'Regions' are defined as the surrounding outside the area of the graph

---

10. This is Euler's Law.

and all enclosed or bounded domains.

E = number of flows of controls (number of edges)

N = number of nodes in the flow graph G.

(5.7) $V(G) = P+1$

Where,

P = number of predicate nodes, contained in the flow graph G. A predicate node represents a decision point in a program.

V(G) provides a quantitative measure of logical difficulty, as it will increase with the number of decision branches and loops. Also, the value *V(G)* has the capability to propose an upper level for module size, which McCabe found as: *V(G)* = 10, meaning that modules with a *V(G)* value of 10 or less are considered well structured and stable.

McCabe's Cyclomatic Complexity measurement was found to correlate with the number of errors existing in a code and the effort required to identify and to repair these errors. There is some evidence that the measure is also correlated with the effort required to develop a module. Productivity decreases in a non-linear fashion as the **density of decision points** increases [Gaf79]; [DeM82]. Variations on this theme were suggested by Chen [Che78]; Basili [Bas79] and Myers [Mye77]. Lately, McCabe and Butler [McC89] suggest the application of Cyclomatic Complexity to architecture of hierarchical design. The Cyclomatic Complexity approach is to measure (and control) the number of sub-trees through the architectural design. The 'design entity' which is analogous to 'number of paths in a source module' is the 'design tree' (the order established by the hierarchical relationship among modules of a system) and the sub-tree. This new direction might be an improvement, but it should be noted that the point in the life cycle where it could be used effectively is very expensive to reach.

McCabe's work was further investigated and modified by researchers, e.g. Myers [Mye77]; Basili [Bas79]; DeMarco [DeM82] and Boehm [Boe82]. But, Shen [She88] for example, criticised the work, and suggests that the Cyclomatic

Complexity is no better than LOC as a predictor of complexity.

However, in spite of the popularity of the Cyclomatic Complexity, two limitations of the approach are of importance. Firstly, it only covers programming effort and has no direct way of dealing with the complexity of the user's (original) problem. Secondly, it does not comes to grips with data complexity.

5.6.4 Composite software complexity

Halstead [Hal72;77] proposed a Software Science Law, which is a composite measure of complexity. His model is based on an assumption rooted in the psychological theory of Miller [Mil56] and Stroud [Str66], that human mind can make only a limited number of mental discriminations per unit of time, the *"human brain follows a more rigid set of rules than it has been aware of"* [Hal77]. Halstead proposed the use of a set of primitive measures that identify the measure of software complexity, at the program level and at the overall level. These measures are:

* Program level.
* Program length.
* Potential minimum volume for an algorithm.
* The actual volume.
* Development effort.
* Development time.
* Project number of faults.

Where the primitives are:

n_1 - number of unique operators appear in a program.
n_2 - number of unique operands appear in a program.
N_1 - total number of operators occurrences in a program.
N_2 - total number of operands occurrences in a program.

The **length** of a program is estimated by the expression:

$$(5.8) \qquad N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

The **program volume** or information volume is defined as:

$$(5.9) \qquad V^* = (N_1 + N_2) * \log_2 (n_1 + n_2)$$

* V varies with programming languages.

The volume ratio is defined as the ratio between the volume of the most compact form of a program and the volume of the actual program:

$$(5.10) \qquad L = 2/n_1 * n_2/N_2$$

The argument is that programming difficulty increases if additional operators are introduced and if an operand is used repetitively

The 'program volume' formulae *(5.9)* which is Halstead's interpretation of a software complexity measurement, is found to correlate with the effort required to create a program. A correlation greater than 0.9 has been reported frequently between these metrics and the number of errors in a program [Cur79].[11] Halstead's Composite Complexity measurement *(5.9)* is used as a method to estimate program size called the Linguistic or the Vocabulary approach [IIT87a].

Halstead's work has received considerable attention and has been subjected to considerable evaluative research.[12] Several authors suggested that the Composite Software is no better than LOC as a measure of complexity, e.g. Kitchenham [Kit87], Shen [She83], Shooman [Sho83]. The argument indicated earlier, that the metrics information being achievable late in the SDLC, is

---

11. These metrics have proven useful in actual practice such as indicating code complexity to programmers.

12. [Fit78]; [Els78]; [Cur79]; [Gaf79]; [DeM82].

applicable for the Cyclomatic Complexity and the Composite Complexity as well. The predicative value of the approach is therefore doubtful. In addition, as a result of incorporating only operators and operands, the method cannot be effectively used in the analysis of 4GL's.

5.6.5 Environmental Composite Complexity

DeMarco [Dem82] differentiates between volume and complexity, as the complexity factors are environment dependent (data; objects; relationships; states and transitions), and not only size dependent. He offers a size correction for the defined Functional Primitives and Complexity Weighting Factors for each suggested category of the Functional Primitives.[13] The volume is evaluated in bits and multiplied by a complexity factor which is based on a decision counts associated with each of the categories. DeMarco's composite complexity is based on Halstead's [Hal77] syntactical complexity and on McCabe's control flow metric [McC76].

Albrecht [Alb83] also approaches complexity as an environmental dependent factor, however, in two different ways. One is by assigning a **relative** complexity factor to each of his function type categories. The second is by assigning a complexity factor to each of these categories based upon system characteristics. The category classification suggested by Albrecht is mainly of a data driven type, while the system compensation factor is basically dependent on the environment.

5.6.6 Inter-connections between system components

Since the modularisation of software, as suggested by Parnas [Par72], has become a dominant concept in building software, metrics have been proposed to assess the

---

13. The categories of Functional Primitives suggested by DeMarco are shown in Table 4.2.

complexity of the inter-connections between parts of a system. The idea is that complexity is related to the proportion of the rest of the system that the programmer has to understand in order to work on his part. This applies to both physical units (e.g. modules) and logical units (e.g. functions). Such metrics were proposed by Belady and Lehman [Bel76]; McClure [McCl78] and Myers [Mye76;78]. Myers has suggested modelling complexity as two aspects, the strength of the module and the coupling between modules. He advocates having as much independence between modules as possible, suggesting that the complexity of the interface between modules is a good predictor of system complexity but has not, as yet, offered an operational definition.

The focus here is quite different from that taken by Halstead or McCabe. Metrics measuring the syntactical constructs or the control flow emphasise the micro-view of an individual program, while interconnectivity takes the macro-level. Although these metrics aim mainly at predicting the maintenance resources, they are of interest as they incorporate a different approach which might be of help in the macro level prediction of resources.

5.6.7 Discussion

However, these deterministic metrics do not suit the objective of this thesis, which is to assess software development complexity at the outset of the development. The reasons are as follows:

- Both McCabe's and Halstead's complexity metrics are essentially code size metrics suitable to assess complexity of completed software products. These metrics can then be used as an analogy based on similarities of project function, or predicting effort required during succeeding cycles.
- Management discipline and technical communication become determinative, while the efficiency and the structure of an algorithm is of secondary importance [McG80]. Thus, for the purpose of this thesis, neither the computational - logical complexity (McCabe's example) nor the

syntactical complexity (Halstead's example), is considered to present a prime cause of difficulties in the Preliminary System Design phase of software development, apart from the natural difficulty that stems from the size of the product. There is no doubt that logical complexity is important at the detailed design, the programming and the testing segments, but not at the outset of the development process, which is the prime concern of this research. In addition, when a computational complexity is identified, it is definable in mathematical terms, while the complexity originated in the socio-psychological arena is not. Therefore, computational complexity can be computerised, measured and dealt with, even though it is often difficult and complex to implement such measurements. It is easier to deal with understandable and measurable phenomenon than with unexplained or unmeasurable ones.

This thesis proposes that the most important complexity determinants are those associated with the interactions among system components, the parties affiliated with the development work, whether they are individual or organisational entities. Interactions among the technical components of a system are important complexity triggers. It is not only in the human societal systems where the Tower of Babel syndrome exists, the 'technical components' of a system do not integrate easily either. Much attention and effort is devoted to these 'technicalities' in order to enable them to communicate smoothly within a system. Hence, two different categories of interactions exist in system development practice and both are important complexity determinants. However, one is measurable in either deterministic or probabilistic form, while the other is not measurable in the deterministic form and it is also questionable whether it is in a probabilistic form. The definition of complexity suggested by [Cur79] is the more appropriate for our purposes.

> *Complexity is a characteristic of the software interface which influences the resources another system will expand or commit while interacting with software* [Cur79].

However, this research argues that it cannot be measured as was proposed by Basili [Bas79a] as quoted in Section 1.8.

The next section analyses the complexity determinants as they are incorporated in the various models for effort estimation. It aims to support the arguments of this thesis as previously expressed and to establish the basis for the complexity approach proposed for the EEM.

5.7 COMPLEXITY DETERMINANTS

There are a variety of factors influencing our ability to estimate the effort required for software development. No one model incorporates all cost drivers that have been identified by the SPEM Esprit Project [Coh88]. Walston and Felix [Wal77] cited 127 different potential factors with a bearing on the successful estimates of effort and schedule. Although the effect of each of these factors cannot be isolated, we could learn what factors dominate the productivity of the software development process.

Indeed, scanning the models represented in Chapter 3 for factors affecting the ability to predict successfully the effort required for building software will lead us to the important factors discussed below and summarised in **Appendix 5A.**[14] Although self evident, there is an agreement as to the significance of this list of factors, it can be easily noticed that similar attributes are phrased or defined differently, which causes subjectivity and thus inconsistency, in their assessment.

---

14. Included in this are factors used by tools which are not described in detail in this thesis. Fuller details can be found in the State of The Art Survey of the MERMAID Esprit project P2046 [Cow89].

5.7.1 User interface and the relative stability of the requirements

Interaction and communication among information technology professionals and the users of the target software play an important role throughout the development. However, early in the development when the problem is being defined, the users' contribution to the process is high and, therefore, necessitates communication. Thus, Walston and Felix's [Wal77] findings are of particular interest for this research. It is important to recognise the high impact on the development process of factors such as user interface complexity and the volatility of the requirements that originated either in the user organisation or were identified during the process of negotiating the appropriate solution. Lack of requirements and stability of design are considered as belonging to the same family of factors. These findings are well supported by research.[15] These attributes are dominating contributors to the **feedback, entropy** and thus, to the **uncertainty** and **complexity** associated with the process of software development.

The implications for the estimation process at the outset of the project life cycle are very clear, the process is bound by uncertainty concerning a great deal of as yet unknown factors.

5.7.2 Management factors: number of decision levels

Noteworthy is the observation by Aron [Aro69] that the number of management levels associated with the development process is a cause for complexity in the software development process. The interactions between systems, in their broader context (as addressed in Chapter 1) are suggested as an indicator for large projects and, therefore, for system complexity. This corresponds with the view that complexity arises in environments which are characterised by rich inter-connections and many levels of hierarchy. The number of decision levels, as well

---

15. Albrecht's underlying assumption is stability of the requirements, Nelson [Nel66], Doty [Dot77], Boehm [Boe81], Jensen and the PRICE-S model use them as modifying factors.

as the number of the project team members, are a source of entropy and thus, should be accounted for in estimating the effort and scheduling of the project. Aron's note that *"the emphasis on management rather than technology represents a major change in the nature of programming since the early 1950's"* [Aro74], indicated a new direction in software management.

5.7.3 Team composition

Although it should be obvious that experience is important for the success of software development, it is worth mentioning the second group of significant factors observed by Walston and Felix [Wal77] which are related to the overall project team composition. Of particular importance are two factors: experience with application type, and with programming language. These findings gained wide support among researchers, as depicted in **Appendix 5A.** Boehm's [Boe81] list of cost drivers supports these findings, in particular, the 'analyst's capabilities'. Freburger and Basili [Fre79] consider this group of factors as one of their two major categories of factors affecting the software development. As discussed in Chapter 2, an error introduced early in software development has a great impact on the process of development, if not detected and corrected early in the process. Experienced staff could reduce the amount of errors in these stages, as well the degree of entropy. In addition, as previously discussed, the PSD is where the proposed solution is negotiated, and whether an understanding of the application domain and the technological opportunities are gained. The benefits from this process cannot be realised if the project team composition is not appropriate. The availability of high quality analysts is a prerequisite for the implementation of the Preliminary System Design. The PSD phase cannot be implemented at all if quality people are not available for it. It should be noted, however, that Miyazaki and Mori [Miy85] did not find this cost driver effective for their environment and eliminated it when calibrating COCOMO. The continuity and stability of the implementation team should be considered also.

5.7.4 Systems interactions

Aron's [Aro69] observations about interactions between systems being the cause of difficulty and thus, of complexity are of interest and importance. The reasons are those previously discussed and associated with entropy, feedback and uncertainty. The behaviour of a system or subsystem in isolation may be very different from its behaviour, when it interacts with other systems or subsystems. Most of the models consider system interactions as adjustment factors for system complexity. However, these factors deserve particular attention, as the potential degree of uncertainty and complexity, stemming from systems interactions, are quite high. Therefore it is crucial to plan for them. Planning might help us in our attempt to cope with the complexity, although it does not secure us from its unpredictable consequences. The dynamics of the real world in which systems are operated are causes for continuous change. Changes in the internal components of a system and/or in the relationships between the systems and the external world occur. Even a small discrete change might give rise to a new unexpected change which in consequence will emerge in the form of multiple changes and interactions at many levels of the system. Noteworthy is the Belady and Lehman [Bel71] study concerning the history of successive releases in a large operating system. They find that the total number of modules increases linearly with release number, but that the number of modules affected increases exponentially with release number. If such changes are not anticipated early and not planned for, the development of the software will run out of control and will need more effort and time than was estimated.

A notion closely related to uncertainty is **variety.** If in doing some task, the variety of information and necessary equipment is large, then the task is complex. Belady and Lehman identify the concept and the attribute of 'largeness' in software development as related to the concept of variety. *"A program is large if it reflects within itself a variety of human interests and activities"* [Bel79b]. Large systems require a group of individuals. Therefore, it is the communication within the implementing organisation, communication between the implementors and the users, the operating organisation, that leads to the emergence of largeness.

Interactions between systems imply variety, hence complexity.

Interconnectivity among systems components strongly affects the effort required for development, in particular at the Preliminary System Design phase where the interconnectivity among systems components are analysed and considered. System interactions should also serve as an indicator for the risk anticipated with the development effort. Abdel-Hamid and Madnick [Abd89] who modelled the dynamics of software development, report that action taken by one subsystem can be traced throughout the entire management system.

5.7.5. Multi-sites development

Software developed in a number of different locations or organisations requires additional co-ordination among the various development teams. Likewise for software developed on different target hosts, which might be in the same or in different locations, or software developed concurrently with hardware. Therefore, these factors are high contributors to the entropy and uncertainty associated with the development process. This stems from the variety of equipment and or organisations affiliated with the process of development. These factors were considered as complexity factors by the early models such as: SDC [Nel66], Doty [Dot77], Aron [Aro69] and adopted by Albrecht [Alb79], PRICE-S Freiman [Frei79] and Jensen [Jen83a] tools.

5.7.6 Re-use of software

Various categories of software routines have their own cost of development, resulting from the degree of difficulty involved. The difficulty and the re-use of software (concept introduced by [Far65]) are the dominating factors for the estimating process at the early design stage [Wol74]. The PRICE-S model considers the novelty of the project as complexity adjustment only if it is a new line of business. Various researches have incorporated the effort needed for re-use

software differently. Boehm [Boe81] did not suppose re-use of software to be a correction factor (which he defines as a cost driver), yet, considers it an important enough factor to provide an algorithm for the adjustment of project profile (size and duration) for re-use of software code. However, his assumption about the effect of re-use being identical between the SDLC phases does not agree with other research, e.g. Black [Bla77] (Boeing model) suggests that the effect of re-used code varies among activities. Boydston [Boy87] suggests the effect of the re-use of software code varies among languages. Walston and Felix [Wal77] took a different approach. They believe that for every percent of new code there will be 3% increase in cost.

Recent work by Pfleeger [Pfl89] has concentrated on re-use of software issues in object oriented software development. They suggest that a cost factor affecting productivity can be defined by the user and, for each factor X, a cost multiplier is generated from estimates of the portion of the project affected by X, the cost of creating X, the cost of incorporating X into the product, and the number of projects over which the costs will be amortised. The results of Pfleeger's model were much better than COCOMO results. They were *PRED (0.25)* of 50% compared with 0% for COCOMO.

Although intuitively re-use of software should reduce the required effort for software development, there is much support for the findings that the required effort for the testing and system integrating segments will increase.

The implication from these findings is the possible shift in the resource allocation among phases of software development as a result of the increasing trend of re-using software components.

5.7.7 Complexity of software product

The COCOMO [Boe81] set of models considers complexity of software product as one of the more significant attributes affecting productivity during the programming effort (it is related mainly to the module level). Walston and Felix [Wal77] consider similar attributes such as overall complexity of code developed

and complexity of application processing, yet, they did not score these attributes highly. Putnam [Put78] accepts that software product complexity affects the development effort. This factor is incorporated in the SLIM tool, although it is not a separate factor. SLIM anticipates only one composite factor, the 'technology factor' which incorporates the major complexity determinants already identified: the technology characteristics of the developer's organisation, the programming support environment, the hardware constraints, the project team composition and the program complexity. Information is not provided on how each of these factors contributes to the technology factor.

If 'software type' is anticipated as an indicator for system complexity then almost all researchers consider it as an important indicator for the required effort, as shown in **Appendix 5A**. However, the grouping into categories varies among the researchers. The terms used are different and even when similar terms were used (real-time; control systems; scientific programs) it is likely that each author interpreted the term differently.

5.7.8 Various size attributes: Data elements, I/O and Files

Farr and Zagorsky's [Far65] model was the first to consider the number of data elements in the target system to be an important driver of development costs. Walston and Felix [Wal77] observed the ratio of number of classes of items in the data-base per 1000 LOC as a factor affecting the programming productivity. Yet, they observed a low productivity change for this factor. Boehm [Boe81] considers the data-base size as a correction factor and defines it similarly to Walston and Felix [Wal77]: the data-base size per LOC.

The emergence of non-LOC sizing metrics expanded the list of complexity determinants with new attributes. Albrecht suggested the measurement of system size by the functions employed, which he found to be highly correlated with productivity of the programming effort. These functions are inputs and outputs, inquiries, internal logical files and external interfaces. Although the SLIM tool uses LOC as its primary size metric, Putnam found that reports or output formats,

files, inputs and number of application subprograms, all correlate with the development effort [Put78].

Albrecht's model [Alb79] was obtained from a data-base composed of business applications. This explains his emphasis on data communication, distributed systems and transaction rates as complexity factors, while in the models originating in the defence and space industries in the US, the emphasis is on attributes such as real-time and time constraints.

DeMarco [DeM82] suggests the data density and the function density of the problem as the composite elements affecting productivity, and hence, as determinants of complexity. He uses the number of Functional Primitives, the number of data tokens associated with each Function Primitive, the number of output elements, number of data elements and the number of interrelationships among the data elements as complexity determinants.

Gaffney [Gaf79] indicates that productivity decreases in a nonlinear fashion as the density of decision points increases. DeMarco [DeM82] agrees with Gaffney's findings for large and monolithic programs, but suggests the complexity of small modules might be better described in terms of an absolute number of decision points.

5.7.9 Factors affecting productivity

The factors which affect productivity have been clarified. However, the various models employ them in different ways. It is yet to be learnt how and to what degree each one of the various factors affects the prediction of each model. However, it is questionable whether the degree of influence of each of these factors can be measured.

Some attempts were made to measure the possible effect of each of the complexity factors on the effort and duration of the software development. Walston and Felix give some indications of the possible affect on the effort required to produce number of LOC per period of time. The productivity assessment models of Aron [Aro69], Doty [Dot77] and Wolverton [Wol74]

categorised the program type according to their assessed level of difficulty. Each category is associated either with productivity rate (LOC per PM) or with cost per category of LOC. The productivity in these models is a composite measure which is based on **composite complexity** assessment. We do not know how the individual factors affect the composite productivity measure.[16]

This thesis does not recommend placing too much meaning on the specific values associated with these attributes, outside the context of the particular environment from which they were obtained. Some of the factors are difficult or even impossible to quantify e.g. human expectations, quality, team synergy. It is impossible to quantify the unique affect of a factor on the effort required for the development, or the impact of a factor on other parts of the system.

Dependent probabilities can be used to measure the effect of each of the factors affecting complexity. But if chosen to do so, an additional question arises about whether the required probabilities could be assessed meaningfully?

Inconsistency in these factors among models and the use of subjective correction factors do not necessarily invalidate the idea of trying to predict effort based on mathematical formulae. But this inconsistency contributes significantly to the imprecision of the resulting estimates. This implies that parametric models may be applied successfully only by analysts who are very familiar with the requirements of the software to be developed, with the assumptions and the context of the sizing, and the resource models.

The state of the art is that there is not yet a well established set of rules or concepts for analysing or evaluating the properties of software systems. This is not to say that measurement of properties of programs and systems cannot be made.

---

16. Walston and Felix article [Wal77] state that **interaction** between the 29 variables which compose their Productivity Index (see Paragraph 3.3.2) are **ignored**. It states, *"this analysis was performed on each variable independently and does not take into account either the possibility that these variables may be correlated, or there may be interrelated effects associated with them"*. But, clearly these variables interact. The importance of each variable was judged by project managers all over the world. But, would the judgement of a manager in US match that of a manager in Saigon? [Fox82] It is a very personal judgement. Therefore, the results of this survey and other similar surveys should be taken into limited consideration, only as an indication that complexity exists and may have an impact the development process.

The problem lies in the fact that the variables which can be measured conveniently do not map readily upon properties or characteristics which are both quantifiable and comparable across a broad spectrum of contexts.

It is interesting to note that models developed prior to the mid '70's emphasised productivity without considering the quality of the resulting product. Although the developers of more recent models are aware of the impact that high quality requirements will have on effort, they do not include quality explicitly.

5.8 THE NEED FOR AN HISTORICAL DATA-BASE

There is a general lack of historical data on completed projects in the public domain. Such data includes detailed characteristics of completed projects and their development environments so that an analogy for a planned project can be identified and, calibration to a new environment can be implemented.

Although articles describing the findings of research have been published, the detailed data-bases of completed and analysed projects are still not generally available. When a data-base is available for commercial use, it is usually dependent on an exchange of information and is, therefore, relatively expensive. The use of such a data-base and a resource estimation model is usually accompanied by the users' commitment to supply the vendor with a post-mortem analysis data of his completed projects. This enables the vendor to characterise the users' profile, to position the user on the industrial productivity trend, and to propose an appropriate technology profile for the user's own environment [Tha88]. This procedure is a means of calibrating an estimation model to the user's environment and as such is a prerequisite to the usage of the model.

5.9 SUMMARY

The chapter has discussed various issues resulting from the discussion in the two previous chapters in this Part. The chapter started by identifying the problems in

current practice of parametric models and continued with an evaluation of the models using empirical studies. The conclusions are that the models yield different results which are mainly attributed to the dynamics of their environments. This implies that the transportability of the models is feasible only if they are calibrated to the target environment. A calibration process needs data from project histories in the target environment, but this is often not available. In addition, and as result of the uncertainty embedded in the process of software development early in life cycle, most of the models do not offer estimates at the outset of project development.

The need for research in the area of resource allocation among phases of development is addressed. Such data could help in providing estimates early in the life cycle. The lack at a public domain of historical data is, therefore, emphasised. The discussion encompassed the importance of understanding the complexity associated with software development. Theoretical and practical issues were discussed and the important complexity determinants were identified.

The chapter closes with a conclusion from Part II in general.

5.10 CONCLUSION - PART II

The lessons learned from the current practice of effort estimation and its implication on modelling the process of effort estimation are now summarised.

The current tools based on parametric models are not widely used outside large software development environments and when used, yield different size estimates for the same project. This phenomenon is attributed to the different links of the estimation tools to the conditions of their development environments. Any variance exhibited by an effort estimation model is not so much due to a difference in perspective among the developers as it is to the dynamic environment that inherently governs the world of software development.

The parametric models require the user to supply values for a wide range of inputs which describe the characteristics of the software product under

development, as well as the environmental characteristics of the development effort. But the required values and environmental characteristics cannot be measured with sufficient rigour to be used in a changing technological environment.

Most of the parametric models are based on the assumption that the product size is known or can be estimated. The size of the software product is measured in Lines of Code (LOC), the Function Point Analysis, or a variation of it. But size is neither known nor easy to establish and, as far as this research is concerned, estimating the product size up-front of the project development, using LOC as the unit of measurement, is not applicable. The Function Point has the advantage of being obtainable earlier than LOC in the life cycle. However, the current practice of function point analysis is not appropriate either, as the Preliminary System Design includes 'satellite' activities which are not directed to the analysis of requirements, but, which require effort.

Most, if not all of, the estimation tools address the development process only from the detailed design stage, after the specifications are established as a baseline. The estimated effort for the Preliminary System Design (requirements and the product design stages) is interpolated from the total effort to the specific phase, using resource distribution among the various phases. None of the models deal explicitly with the Preliminary System Design phase of software development.

There is a pattern of resource allocation between the Preliminary System Design phase and the reminder of the SDLC. This pattern could assist in extrapolating the total effort required for the software development from the estimated effort for the PSD.

A trend can be observed towards the development of knowledge-based techniques for supporting the sizing and estimating of the software development effort. One area is the use of the analogy approach for the identification of similarities with previous projects from which we could learn about the future project [Cow88].

An Expert System as an aid for the calibration process is suggested by Cuelenaere et al. [Cue87]. They calibrated the PRICE-S for their environment by using an Expert System as the interface between the estimator and the tool.

In Chapter 2 the processes and the objectives of the software development and software estimation were analysed. That chapter culminated with the perception of the future trends of software development and the belief (of the author of this thesis) that various methods of effort estimation are required to support the different decision processes addressed in each of the software development phases. An analysis of the primary suggestions for building an Effort Estimation Model was offered. Two basic concepts were suggested, a base model for effort estimation and phased base estimation process.[17] Chapters 3 and 4 introduced the various models and tools currently available for estimation of the project size and effort. The analysis in these chapters, the critique of the current practice in this chapter and the introduction of the issues associated with complexity brought the conclusion that the current code of practice supports neither of the concepts recommended in Chapter 2.

Part III will focus on the Effort Estimation Model (EEM) the methodology and the assumptions it is based upon.

---

17. See Paragraphs 2.8.2 and 2.8.3.

PART III

THE EFFORT ESTIMATION MODEL (EEM)

Part I focused on the problem domain, the estimator dilemma, the process of building software and that of estimating it were analysed. Part II presented an analysis of the state-of-the-art in sizing software products and estimating the effort needed for developing them. The difficulties associated with the estimation process were indicated and the limited understanding of what these models measure and represent was recognised. Each part culminated with the implications for the Effort Estimation Model.

Four chapters encompass Part III. Chapter 6 discusses the methodology used in building the Effort Estimation Model and the methodologies incorporated in the model. Chapter 7 focuses on the Effort Estimation Model itself. The infrastructure for the knowledge-base for effort estimation as well as for further research is described and analysed. The fundamentals of the EEM are discussed, the EEM is presented and demonstrated in detail, along with examples and design features (data models and function charts). A case study closes this chapter. In chapter 8 the EEM is evaluated using qualitative and quantitative approaches. The last chapter of this thesis summarises the major principle upon which the EEM is built and the advantage of the approach taken. The discussion culminates with the contribution for further research.

Chapter 6

RESEARCH METHOD

6.1 INTRODUCTION

This chapter focuses on the research methods used to build the EEM. The approach utilised in modelling the EEM, the methods employed in the knowledge acquisition and the data collection considerations will be discussed.

6.2 DEVELOPMENT METHOD

Two approaches for modelling are recognised in this thesis, the analytical (structural) model and the empirical (descriptive). In using the analytical approach, a system is described in terms of how it is presumed to work, and predictions about its behaviour are derived from measurements or predictions about the known behaviour of its components. The empirical model does not attempt to model the underlying structure. Predictions about the system behaviour are made by extrapolation from previous observations relating the various inputs to the system behaviour. Structural models are considered appropriate for small systems, as the investigator can then make reasonable hypotheses about how they work. Descriptive models are more plausible for large systems in which the underlying structure is not well understood, but nonetheless

important so that some predictions can be made [Bro83].

Generally, software development cannot be considered a small task, even when the 'underlying structure' of the development process is understood. This statement may seem contradictory in relation to the discussion in previous chapters and, therefore, deserves an explanation. The pillars of the process are the activities and they are well understood. Their routeing, although changeable, is also generally understood and known for each of the common strategies.[1] What is not well understood is the complexity associated with the process of software development and which is caused by the uncertainty inherent in this process.

By the same arguments used to justify the conceptual structure of the EEM, the analytical approach was chosen as the main approach for building the EEM. The descriptive modelling approach is incorporated in the EEM for predicting planning approximations where the underlying structure is not clear at the outset of the project.

6.2.1 Conceptual design of the Knowledge-base

It is appropriate to start the process of developing the EEM with the design of the knowledge-base. The first major task is the identification and determination of fundamental properties of the model.

The development of this part was based on the author's own professional knowledge, supported by the some of the concepts used in Method/1 [And79]. The activities included in the design are:

- Segmentation of the SDLC for each of the strategies into a Work Breakdown Structure (WBS). The parts encompassing the WBS must be manageable, and must present the milestones for the development and have identified deliverables which are often prerequisites for continuing the development process.

---

1. The reader is referred to Section 2.2 and Paragraphs 7.4.1 and 7.4.2 of this thesis.

- Identification of the major contributors to the cost associated with each activity involved in the alternative strategies. These are the 'cost drivers' used by the EEM.
- Assignment of Standards of Effort (SOE) units to all of the combined entities, each composed of an activity and an associated cost driver. Two procedures were associated with this process. The first was based on statistical information from projects' histories in a variety of environments. The second was a fine tuning process based on the knowledge gained by walkthrough sessions and the author's judgement resulting from analysing the data collected with questionnaires. This has taken place in the latter stages of the design process.
- Identification of rules which allow the assessment of complexity and risk associated with a software project.
- Identification of rules which allow adjustment and calibration of the effort required for software development for different technologies are used and/or when the particular development is state-of-the-art.

A questionnaire was developed. This was intended to explain the concepts of the EEM, to exchange ideas about the validity of the concept and the feasibility of its implementation, to elicit additional knowledge as well as to collect actual and estimated data. A copy of the questionnaire can be found in **Appendix 6A.**

6.2.2 Knowledge acquisition

Two pilot cases were conducted, one in Israel and one in the UK, each of which included a number of projects and a few individual walkthrough sessions, from which considerable experience was gained which improved the questionnaire and hence the EEM.

The first case study aimed mainly to clarify the concept, the feasibility and the data collection process. It was held in Israel and a Hebrew version of the questionnaire was used. Project managers, senior project leaders and a manager

of an information technology unit participated. The ideas, concepts and assumptions were presented to them. Each of the attendees tried to complete a questionnaire. This was followed by a discussion of misunderstood issues, a critique, suggestions for additional topics to be covered, and the difficulties which were anticipated in completing the questionnaire. The conclusions from this first walkthrough were:

- The validity of the concept for application development was accepted, although a concern was expressed as to the accuracy of the relevant data at the outset of project life cycle.
- The Validity of the concept, as it is, for the development of basic system software such application generators, was questionable.
- Ambiguity and vagueness of some of the questions were identified. For example, it was not clear what a 'major report' meant. However, an exact definition would not be of help at the outset of project, where only a list of reports exists. There is a need here for the analyst's experience and judgement. Therefore, for the purpose of this thesis, a decision was taken that the analyst should apply his judgement following some guidance given by the EEM. Similarly, the meaning of a 'screen' was not clear. Screens differ not only when used for various purposes, (e.g. inquiries, menus or data entry screens, all of which are covered by the questionnaire) but also among hardware used. For example, an IBM definition of a screen differs from the NCR definition.

Based on this discussion, the first English version of the questionnaire was developed. The second case study was held in the UK with the same objectives as the first case study. Yet, the procedure was different. Three senior project managers, who were engaged in three different projects, received the questionnaire from their information technology manager. They were asked to complete the questionnaires and to comment in particular, on the way the questions were phrased (and whatever else they thought was worth commenting on). The three projects were chosen so that the span of applications and

development strategies were as wide as possible. It was important to check the feasibility of the concept for a variety of development strategies and project types. Therefore, the selection included two projects which were developed using different strategies. One used the customised strategy and the second used the application package strategy. The third project selected was a novel project, concerned with the introduction of a new office automation system into an organisation. The three completed questionnaires were returned accompanied with a detailed covering letter including the project leaders' comments, identification of ambiguities in a few questions, and suggestions for phrasing them differently or adding further explanations.

Individual walkthrough sessions were conducted with a few practitioners from variety of organisations such as software houses, software contractors within governmental organisation and within the private sector.

Corrections to the questionnaire were made to enable the data collection process.

6.2.3 Data collection

The data collection and the walkthrough process aimed to:

- Validate the method used by the EEM.
- Analyse the estimates it yielded.
- Identify rules for corrective action when different technologies are used.

Therefore, collecting data from a variety of organisations, and from variety of projects, was considered an important goal. The possibility of establishing a follow-up walkthrough session with members of the organisation from which additional knowledge could be acquired was considered an advantage. The decision to follow this data collection process was taken, although it was clear that this procedure jeopardises the homogeneity of the experiment.

Accordingly, questionnaires were sent to individuals who were introduced

to the research through intermediate facilitators. The individuals were chosen based on their expertise and their involvement in managing software projects as well as their interest and willingness to take part in this research. A walkthrough process accompanied most of the cases, either before or after the completion of the questionnaires.

6.3 BUILDING THE PROTOTYPE

Based on the principles and the concepts discussed in Chapter 7, a prototype was built using a PC-based expert system shell PESYS[2]. The questionnaire, the function chart and the data models described in Chapter 7 established the functional and the technical specification as well as the design from the view-point of the user.

The following aims directed the building of the prototype:

- To demonstrate the EEM and to evaluate it.
- To learn from the process of using the EEM, in semi-real world.
- To collect data and to establish the basis for an historical data-base.
- To evaluate the use of the Expert System technique.

An expert system shell was used to take advantage of the reasoning facilities it offered. It was hoped that it would facilitate the tracing and allow to exhibit easily the assumptions and decisions taken throughout an estimation session, a benefit that was well presented in practice. However, quite early in the prototype development process it became clear that the EEM could not benefit from all the facilities offered by the shell, e.g. the 'what if' facility due to a shortage in memory space. Hence, part of the model was re-developed using Turbo Pascal.

---

2. PESYS is a rule base system developed as a part of the doctoral thesis by Edgar A. Whitley [Whi90], in the Information Systems Department of the London School of Economics and Political Science.

Chapter 7

THE EFFORT ESTIMATION MODEL (EEM)

7.1 INTRODUCTION

Armed with some understanding of the current practice in estimating the size and cost of software projects, it is now easier to suggest an alternative to assist the software manager with the estimation dilemma. This chapter describes the Effort Estimation Model (EEM). The discussion opens by re-emphasising the objectives of the estimation process at the outset of the project life cycle and with an overview of the fundamentals of the Effort Estimation Model. The structure of the EEM, the knowledge-base used and design features such as data models and function charts are presented. The effort estimation process, the functions and the features incorporated in the model are demonstrated in a number of examples. The chapter closes with two case studies.

The fundamentals of EEM stem directly from the analysis in previous chapters, they are:

* Decomposing the problem.
* Estimating the process from bottom-up.
* Recording of assumptions and decisions.
* Applying size metrics for the Preliminary System Design.

7.2 THE FUNDAMENTALS OF THE EFFORT ESTIMATION MODEL (EEM)

This section focuses on establishing the framework of objectives to be incorporated in the Effort Estimation Model (EEM).

The objectives of estimation process at the outset of project life cycle (at the Project Planning Phase) are to develop budgetary estimates for the Preliminary System Design Phase (PSD) and to provide coarse estimates of the effort required for the total development. Decisions associated with development strategies might affect the effort required, while the estimated effort might have implications which affect both the chosen strategy for development and the functionality of the proposed solution. An important part of the decision process at the outset of the project life cycle is associated with the feasibility of the suggested solution and its foreseen costs.

The assessed complexity and risk associated with software development provide an insight into the processes that affect costs and resources. Thus, knowing the estimated effort and the perceived complexity and risk at the outset of the project supports the process of evaluation of alternatives, enables the project manager to plan resources, and allows their scheduling when needed. Such a plan ensures cost and schedule visualisation of the process, as well as (technical) performance measurements of the emerging product. Hence, the estimated effort required for the development of a software product, and/or for the adaptation of a software package is an important component of the economic evaluation of a proposed project and its feasibility. The EEM estimates include the effort required for the development process by all parts of the organisation, the contractor as well as the user organisation. These estimates support the decision processes associated with the overall management of software development, they facilitate management involvement and thus are considered critical success factors (CSF) for the management of a software project.

What is missing? The previous discussion and analysis led us to the understanding that each process for developing a software system, models the complex environment it aims to serve, and includes an implicit model of itself. This characteristic has implications for the ways in which we could assist software

managers in their estimation dilemma. The first is that the process of estimating should be an **evolutionary, iterative and interactive** process, with a built-in **feedback mechanism**. The second implication is that a **closed algorithm cannot be a satisfactory solution** for an effort estimation model at the outset of a project.

Much of the difficulty in estimating software effort arises from the degree of uncertainty associated with both the application domain and with the dynamics of the environments in which the software is being developed and that it is aiming to serve. Each of the software development phases and/or segments is based on its predecessor, which can be viewed as its specification and could be formalised to help in reducing the uncertainty associated in its implementation. However, this is not the situation for the first phase of the software development, the Project Planning. The only representation of the problem existing here is a verbal expression of needs, which may themselves not always be known. The uncertainty in software development and in estimating the effort required for its development, is a consequence of the nature of the real world and the essential need (and difficulty) to abstract from that world. *"Imprecision and incompleteness of models on which this process must be based implies embedded absolute uncertainty"*, [Leh89]. This uncertainty is why most Parametric Models form estimates only after the a major part of the Preliminary System Design is completed. Some of these models estimate the effort required for the PSD phase by interpolating them from the estimates of the total effort.

What is the EEM all about? This thesis limits its scope to establishing the concept and the design of a detailed method for estimating the effort required for the Preliminary System Design phase. The assumption is that it is possible to **estimate** the effort required for the PSD quite accurately. However, at the same time only **coarse estimates** of the effort needed for the total development are feasible. These coarse estimates can be obtained by extrapolating the estimates for the first phase (PSD), and the likely distribution of effort among the development phases, which is known statistically from project histories, and by judgement about the foreseen effort allocation in the particular project.

An estimate is developed for a single phase of a project when there is a reasonably precise definition of the scope and objectives of the software to be

developed. A coarse estimate is a forecast of the total effort for the systems development including the Construction phase. They are developed early in the life cycle before the completion of the PSD, when uncertainty about the problem and the solution to it is still high. Therefore, measurement and judgement are integrated in the process. The forecasts and estimates are updated as the project proceeds, when new information is gained as shown in Figure 2.4. It is customary to assume that as the development process proceeds the uncertainty associated with the development is reduced. However, this is not always true as new uncertainties may be introduced which should be incorporated into the updated estimates.

The importance of the PSD phase, and its critical to the welfare of the project and to the achievement of the goals of software management, led to the consideration of this approach. It should be noted that the EEM is a supplementary model which provides a method for effort estimation at the outset of the project life cycle. Ideally it should be applied in tandem with other models which support the estimates for the later stages.

7.2.1 Decomposition of the problem

The estimation of the effort required for a software project is viewed as an 'ill defined' problem, in particular at the outset of the project life cycle. It is a unique decision making activity which is a form of problem solving, and in most cases the problem to be solved is too complex to be considered in one step. Decomposition techniques are a natural approach to problem solving. If the problem to be solved is 'ill defined' and too complicated to be solved as one unit, it can be subdivided until manageable problems are encountered. Each problem is solved in isolation and the solutions are combined to answer the original problem. Effort estimation is a complicated decision process. It is a multi-attribute and multi-party decision process. Thus, the decision process associated with effort estimation as a whole should be decomposed into parts. Each development phase is a unique decision process and, therefore, should be based on different parameters and tools.

Decomposing the decision process associated with effort estimation, and basing each decision on size metrics representing the effort associated with the activities incorporated in the particular phase of the process, enable us to transform the process from a 'black art' into a series of steps. Such an approach may help in providing estimates within an acceptable degree of risk. Although there is no panacea to the estimator's dilemma, a common framework promoting a common culture is an essential step forward. A systematic approach helps the estimator to judge better. This research takes the view that the two concepts described previously, the base model for effort estimation process and the phase based estimation process, provide a solution to the problems stated.[1] A base model supports the requirement for a systematic approach to the effort estimation task. The phase based estimation process reduces the degree of uncertainty associated with that process. The infrastructure for a base model for effort estimation proposed in this thesis could be only considered with the bottom-up approach is being implemented.

7.2.2 Recording and tracing assumptions and decisions

The uncertainty associated with the software development process, its iterative and evolutionary nature, mean that assumptions taken throughout the software development process will change, as will the decisions which are based on them. Nevertheless, these assumptions are the basis for the estimation process. It was already noted that underestimating the effort is caused by the common phenomena of 'short-term memory' and, of more importance, by the reliance on the knowledge and the memory of individuals who might not be available when their expertise and knowledge are needed.[2] Therefore, the EEM addresses this issue by recording the decisions taken throughout the software development and

---

1. See Paragraphs 2.8.2 and 2.8.3.

2. See Paragraph 1.8.2

the effort estimation processes. Decisions and assumptions associated with the project include:

- Development strategies, e.g. the use of software packages, the use of development tools, or the use of prototyping.
- Work break-down for a particular development process.
- Software size metrics and their values.
- Perception of software product complexity, as well as other types of complexities which are associated with the development process, i.e. organisational, technical and project team complexity.

Hence, a way which allows us to **trace the assumptions** taken while estimating the effort, brings us a step forward in our aim of improving the estimation procedures and of understanding deviations from the basic assumptions upon which the estimation process was developed.

Recording the assumptions and decisions taken throughout the estimation processes and linking them to the relevant activity or segment, enables us to incorporate a feedback mechanism into the process of software development. Such a mechanism could point out specified deviations from the basic assumptions and suggest corrective measures for the estimated effort. Tracing the assumptions enables us to reason about the decisions taken throughout the processes of our concern. However, the relative importance and the value of information stemming from recording each decision should be considered.

7.2.3 The applicable size metrics for the Preliminary System Design

Each phase in the SDLC addresses various issues in the development process and employs activities characterised by various attributes. It is thus obvious that the size metrics associated with each of these phases will vary. The best representative unit of measure for the effort required to implement a phase of software is based on the input unit of measure to this phase. Early in the software development the

focus is on the functionality of the target system, therefore, the most appropriate size metric is 'function' oriented.

The proposal is to decompose the software development life cycle into phases, segments and activities, to identify size components which are the major cost contributors to the PSD work, and to associate with each component and activity a 'standard of effort' measured in person hours. However, the variety of projects and their associated environments, and the introduction of new technologies, quite often imply that this is not applicable for all activities. Some activities vary widely from one project to another and thus their 'standards of effort' are not known *a priori*. The effort required for these activities should be estimated separately by the effort estimator for each project, using his experience and expert judgement. Standards measurement of project history is mandatory, but measurement must be used in conjunction with judgement.

However, estimates cannot be better than their ingredients. Based on the information available at the Project Planning phase, only coarse estimates of the effort for the total project can be provided. Although these estimates will not remove the uncertainty, they will almost surely place the organisation in a better position, to deal with the unknowns and to take advantages of developments as they occur. Knowing ahead of time where the trouble is going to come from, will make some difference.

7.3 THE STRUCTURE OF THE EFFORT ESTIMATION MODEL (EEM)

The Effort Estimation Model for Software Development Projects is a support method for the estimation process, which takes place when a project is about to commence and uses attributes and measurements consistent with the level of knowledge generally available in the organisation at that time.

The effort estimating task relies heavily on the judgement of experienced performers. Therefore, the EEM is built as an interactive process enabling the estimator to interact with it as though it were the expert, thus, providing professional assistance when needed. The EEM recommends an activity

breakdown for alternative development strategies, provides 'standards of effort' for various size components that generate the cost of the activities to which it contributes and assists in identifying and assessing complexity and risk. Furthermore, at a later stage, it provides guidance for choosing the appropriate model for effort estimation based on the estimation objectives, type of system, and also prepares a schedule to the project. By taking an advantage of AI, it should 'learn' from the data generated by active projects and incorporate that experience into the knowledge-base.

Consequently, the EEM has the following 2 main components (as shown in Figure 7.1):

- **An interactive component** which enables the estimator to converse with the model throughout the estimation session, to use his judgement when corrective action is needed, and to interrupt when additional information is required. The interactive component make use of a series of questionnaires with the aim of directing the estimator through the estimation session and to capture the estimator's assumptions and decisions made throughout the estimation session for further processing. It is also the means by which the system can provide an explanation of its reasoning.
- **A knowledge-base** and the inference engine, and the reasoning facility which examines and uses the knowledge-base. The knowledge-base is a store representation of the expert knowledge, and includes three elements, namely:

  * A base model for effort estimation. The major element in the base model is the Life cycle decomposition into phases, segments and activities and their associated affiliated information.
  * Complexity and risk assessment rules.
  * A data-base of estimation models.

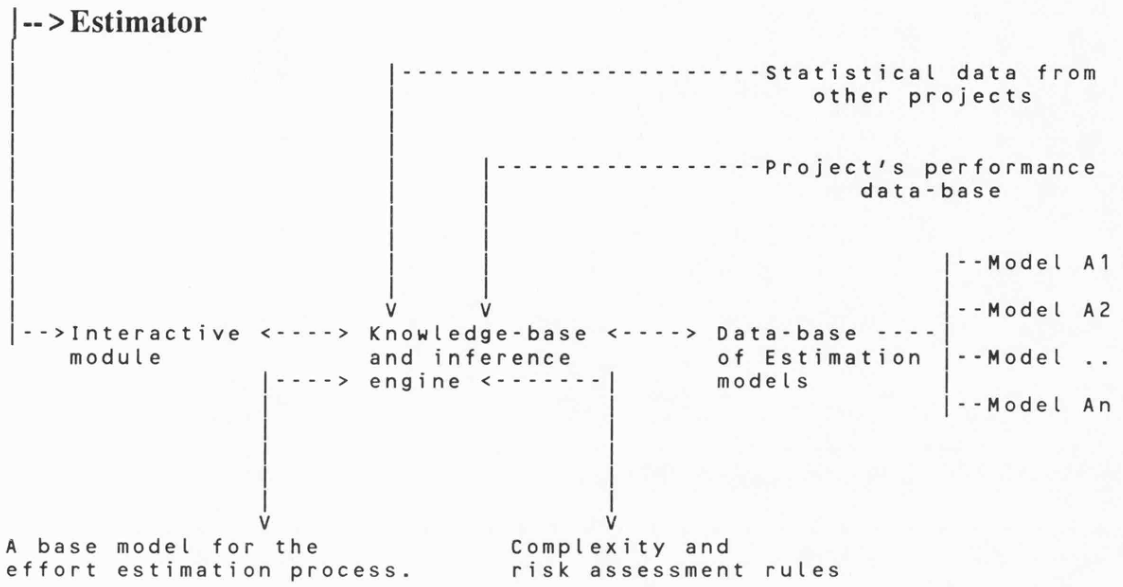


Figure 7.1 The conceptual model of the EEM

7.4 BASE MODEL FOR EFFORT ESTIMATION

7.4.1 The Software Development Life cycle

The Effort Estimation Model (EEM) assumes the use of a systematic management framework and that the project development process is supported by a standard software development process which serves as a tool for directing the creation of new information systems. It is customary to consider the software development process as having a life cycle, characterised by a top-down approach of breaking the process into manageable, logical and functional units. The development process is decomposed into phases, each having defined starting and ending points. Each phase is further decomposed into individual work segments, each of which produces pre-defined end-products and aims to achieve a specific target. Each segment contains a group of standard controllable activities. The EEM uses a consistent set of terms to describe this conceptual top-down approach:

- A **phase** is a major self-contained component. Four phases encompass the development process:

  * Project Planning.
  * Preliminary System Design.
  * Construction (Detailed Design and Implementation).
  * Operation.

- A **segment** is a logical part in accomplishing the objectives of a particular phase. The Preliminary System Design phase includes the following segments:

  * Organisation.
  * User requirements.
  * User design.
  * Technical design.
  * Technical support.
  * Construction schedule.
  * Cost / benefit analysis.
  * Management review and approval.

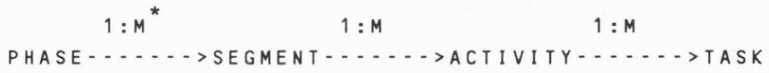
If hardware and/or application software procurement is considered as part of the project, then the following segments will be added to the PSD.

  * Hardware and software direction.
  * Application software evaluation and design.
  * Hardware and software selection.

- Each segment contains a group of standard **activities** which provides the project team with guidelines to accomplish the segment's end result. For example, the Project Definition segment of the Project Planning phase contains activities such as:

* Initiate a project.
* Review present status.
* Identify business objectives and information strategies.
* Survey information need.
* Identify hardware and software environment.
* Develop conceptual design.
* Investigate application software alternatives.
* Evaluate development alternatives.
* Prepare project impact analysis.
* Prepare project plan.

- Each activity is decomposed into individual **tasks** needed to perform a specific activity. For example, the 'Prepare project plan' activity includes a task in which the effort estimates for the next phase are established.



1:M
* The -------> notation means, a phase is decomposed into one or more segments, a segment is decomposed into one or more activities, and activity into one or more tasks.

The basic component common to all frameworks for software development is the **activity**, to which a satellite of attributes are associated. These attributes are, for example, the objectives for implementing the activity; the intermediate product deliverables which indicate the state of the activity; the cost drivers which identify the major contributors to the effort associated with the specific activity; the 'standards of effort' associated with a cost driver and an activity (see Paragraphs 7.4.3 and 7.4.4); the outlines of recommended documentation; the prerequisite and the dependable activities. All types of activities share common information and have similar notions of causality, time relationships, and milestones. A partial view of an activity schema is shown in Figure 7.2. The base model represents activities of a range of particular software processes and allows reasoning about their use.

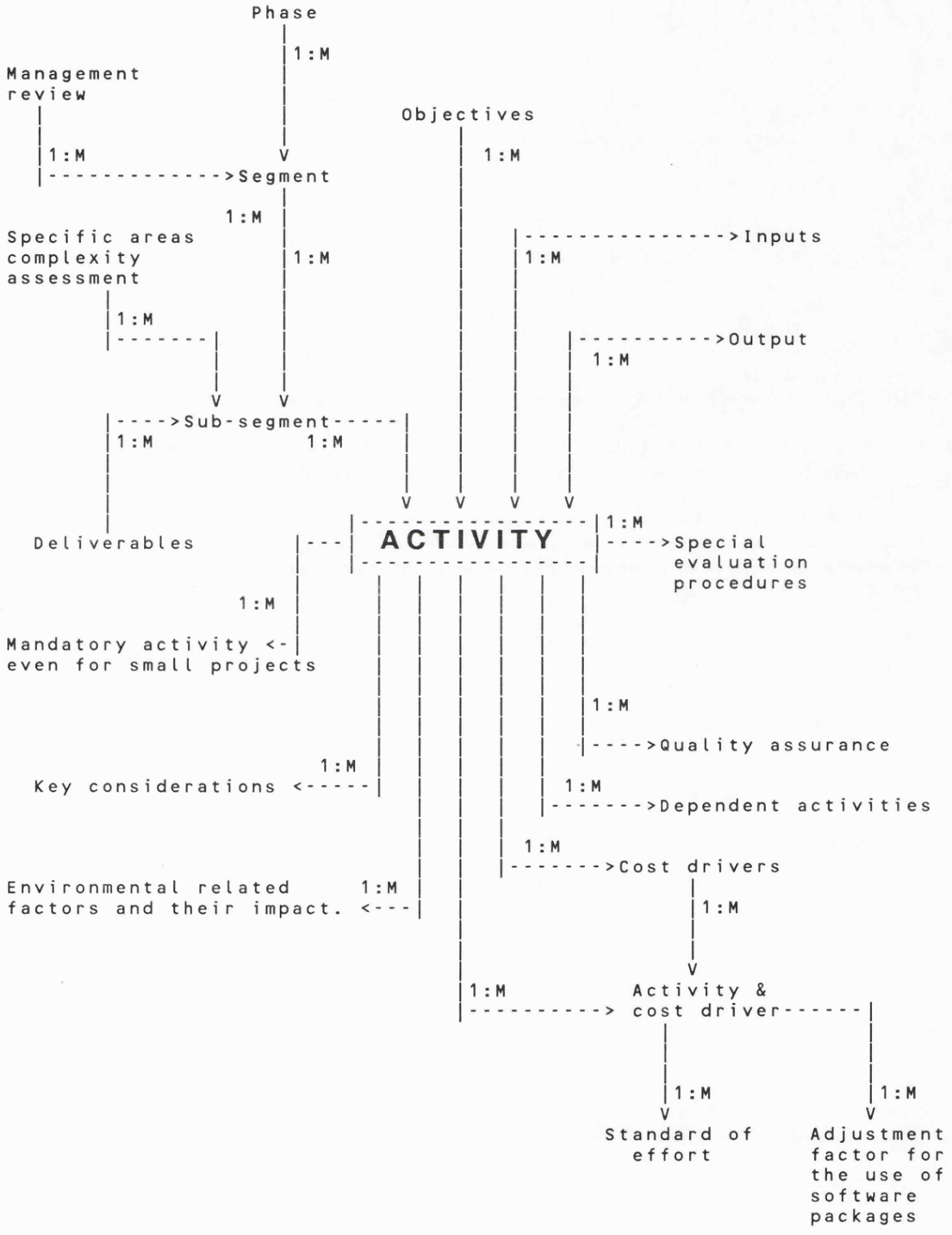


Figure 7.2 An activity data-model (a partial view)

These phases and activities portray the software development life cycle as viewed by the user and provide the basis for effective management. The set of activities and segments (groups of activities) may overlap in time but each must be scheduled for completion prior to a dependent sub-segment or an activity.

Definition of the software life cycle phases

The process of software development and the various paradigms currently in practice, were addressed in Section 2.2. The objectives, functions and main concerns of each of the phases were analysed and presented in comparison to Boehm's traditional life cycle process. Here, the complementary focus is on the view of this thesis regarding the processes associated with each of the phases.

Phase 1: Project Planning. The Project Planning addresses both project definition and feasibility issues. In the project definition segment the preferred concept for the software project is stated, the software development strategies are formulated and the superiority of the chosen concept over alternatives is presented. The Project Planning phase accepts the general needs or problems as inputs and proposes a comprehensive scope, an agreement on problems and a definition of a project, which includes a work plan for the next phase. The deliverables of this phase also include the following four articles: the system overview, the overall strategies for the target system, the functions to be incorporated in a given project, and the data-model to support them. The Project Planning phase is implemented by looking into fact gathering and analysis, interviews and discussions.

Phase 2: Preliminary System Design. The objectives of this phase are to determine how the target system should be implemented to meet the business needs of the organisation and to obtain the commitment of the management to the proposed system, before the major portion of the project development cost is incurred. The main concerns of this phase lie in the following issues:

* What does the system do from the user's viewpoint?
* How does the system operate from a technical viewpoint?
* What are the estimated operating costs and benefits of the system?
* What are the estimated installation costs and time-table?

This phase includes the 'specification requirements' and 'product design'. It also includes complete and validated:

* Functional and technical specification of the 'user requirements'.
* Design from the user point of view.
* Interface, performance, security and control requirements of the software product.
* A complete and verified specification of the the overall hardware and software architecture and the project data models.

Appendix 7A contains the decomposition of the PSD phase into segments and activities, as defined and used in the EEM.[3]

Phase 3: Construction or Detailed Design and Implementation. The objectives of this phase are to finalise the system design and install successfully the system in the operational environment of the company. The phase includes the detailed design, code and debug, test and preparations. The objectives of this phase should be accomplished with:

* Developed procedures.
* Trained users.
* Ensured acceptance of the system by both computer operation and user personnel.

Some development processes employ all the activities included in the

---

3. The list of activities is adopted from Arthur Andersen Method/1 [And79].

conceptual framework whereas others may employ only a subset of the activities for a particular project or, they may find it necessary to add activities to the work breakdown. The EEM allows the user to delete unnecessary activities. It is also possible to add the effort required for activities which are not specified in the proposed work breakdown, or for activities which vary highly among environments.

7.4.2 Alternative strategies for software development

The EEM currently recognises four alternative strategies for the software development process as shown in Figure 7.3. The first three strategies follow the Waterfall model. The fourth is geared toward a fluid environment with changing business needs as well as changing organisational infrastructure, and provides a flexible enough route to allow changes in requirements to be defined and obsolete functions to be eliminated.

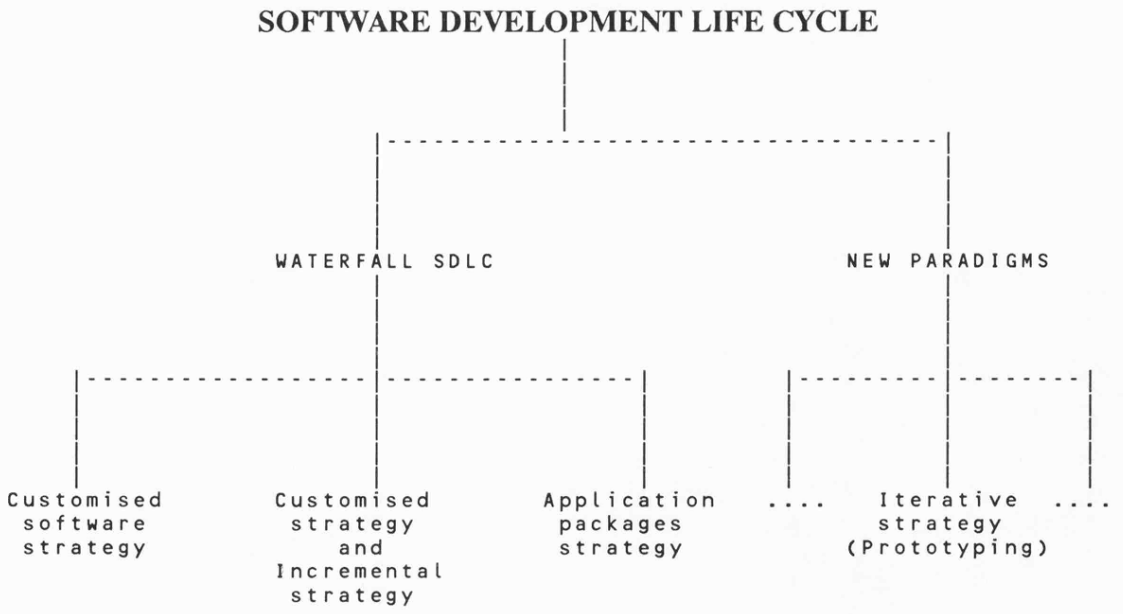


Figure 7.3 Alternative strategies for software development used by the EEM

The Customised software development follows the Waterfall SDLC model, but also allows the integration of the prototyping paradigm into the classic flow of project development using the EEM life cycle decomposition into segments and activities, as shown in **Appendices 7A** and **7A1.** It thereby improves both the complex communication and feasibility decisions involved in the process.

Customised software strategy and Incremental development. The Incremental development as opposed to the 'pure' Waterfall model concentrates on short-term results. Project development is partitioned into increments, whose development is scheduled or phased over the total development cycle. Each increment is a subset of the planned software product and provides specified system functions. The emphasis here is on overall planning of the software product, yet, the implementation is partitioned. Each increment is estimated for its required effort and schedule and is managed separately. The planning and estimating processes of the short-term budget are implemented for the deliverables planned in the intermediate future. The evolutionary delivery process employs the management of relatively small projects. Taking the evolutionary approach it becomes easier to control each increment, to operate to a stable plan, thus significantly decreasing the volatility of the requirements. However, by controlling this factor we reduce the 'overall' system functionality as additional functions requested by the users become a planned part of the next increment.

Application packages from many different industries are readily available and may save the developer much time because much of the time-consuming and costly ground work has already been done. One should not reinvent the wheel every time there is a need for a piece of software. This strategy also follows the Waterfall SDLC model. It is a sub-set of the customised approach for software development. When this strategy is taken, the SDLC is changed and includes some additional segments while others might be omitted.

Iterative development. The new paradigms result from the new technological opportunities that have been developed since the introduction of the Waterfall model. The emergence of fourth generation languages and productivity tools for end-user computing has brought the need for a systems development approach very different from the conventional one. This route varies distinctly

from the customised and application packages strategies in that it allows for a 'trial and error' approach to problem solving, when a specific solution is impossible to identify initially.

When this strategy is chosen, the activity list decreases. However, there is no guarantee that the effort required for the entire system will be less than if implementing the same system using the custom-made approach. Even so the time and cost of the first iteration may be substantially less.

7.4.3 Cost drivers

The EEM assumes that each activity is associated with a standard list of cost drivers involved in the process. The cost drivers are size attributes of the project.[4] The cost drivers serve as the basis for estimating tasks associated with each activity, during the estimation process. A cost driver is, for example, a transaction, an input document, a report, a screen, a contract to be signed, a software package for final evaluation, a modification of a software package, a request for proposal or a project team member.

Some of the cost drivers identify an overhead for a system and some stem from the need to motivate and train the project team. Thus, a project team is identified as a cost driver for the organisational set of activities, although, the number of team members assigned to a project is known only as a result of the estimates. A cost driver might be associated with one or more activities.
Appendix 7B contains the list of cost drivers used in the EEM.

---

4. The term **project size** and not **product size** is used here, since this metric includes size attributes of three different sources: the target system, the replacement system and the process of implementing the PSD. These will affect the effort required for the total development, however, they are not **product attributes** in the common use of the term

7.4.4 Standard of Effort

A Standard of Effort (SOE) is the organisational inverse of a standard rate of productivity, for either the amount of work required to accomplish one work unit, or for a defined cost driver. 'Standard of effort' multipliers, which are not expressed in work units are, for example, a project team member or the system overhead. The 'standard of effort' which is associated with each of the cost drivers and correlated activity may differ between environments for an identical cost driver and associated activity, according to the **complexity** of the project. Therefore, the EEM associates each combination of activity and cost driver with one of three different 'standards of effort' according to the assumed complexity levels of the system: simple, moderate or complex. The degree of complexity is a subjective classification since human beings are involved in the development and thus in the complexity assessment processes. The various parties involved in the development and the complexity assessment processes may differ in their productivity level and in their attitude and understanding of the project under discussion. The same parties might also have conflicting objectives. A 'standard of effort' (SOE) will be of either a direct or indirect type. An example of an indirect type is the management and administration effort.

The partial structures of the base model used in the EEM are shown in each of the Figures 7.4 - 7.7, each of which includes an additional part of the conceptual scheme for the base model or the project's view.

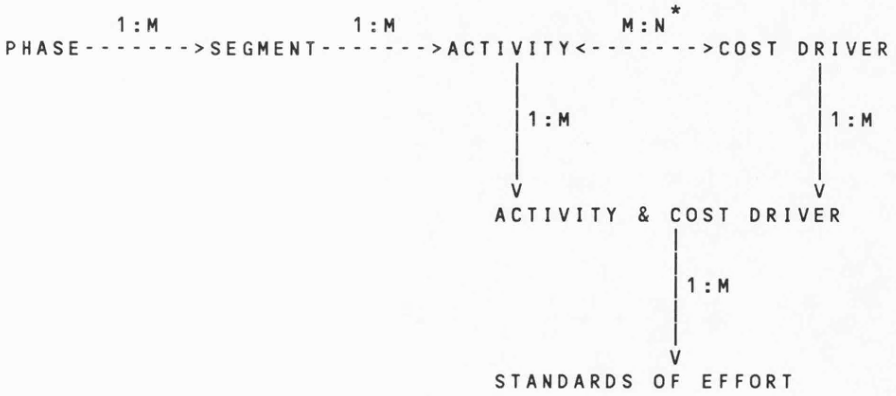


Figure 7.4 A partial view (a) of the conceptual data-model used in the EEM (A base model view)

*The M:N <------> notation means that each activity is associated with zero or more cost drivers, and each cost driver is associated with zero or more activities.

However, some activities will vary in the effort required depending on the number of users, potential availability of software, expected technical complexity, etc. The 'standard of effort' is, therefore, not known for each of the activities involved in software development. Hence, estimation of the effort required to accomplish these activities requires the judgement of the estimator. A situation in which an 'average' effort will not be of great help is for example the first development of a communications-based system in an organisation which is incompatible to any such process developed elsewhere. Such a process should be decomposed into its detailed activities, each activity should be evaluated and estimated separately. The only help a general model could offer for such a process is in a general breakdown into activities and some very coarse ranges of effort.

Figure 7.5 A partial view (b) of the conceptual data-model used in the EEM (A project view)

It is therefore clear that estimates concerning a particular phase, cannot be developed unless this phase is clearly described and understood by its users. This research addresses the effort estimation issue with regard to a single system. The conceptual view of the EEM will take form as shown in Figure 7.6.

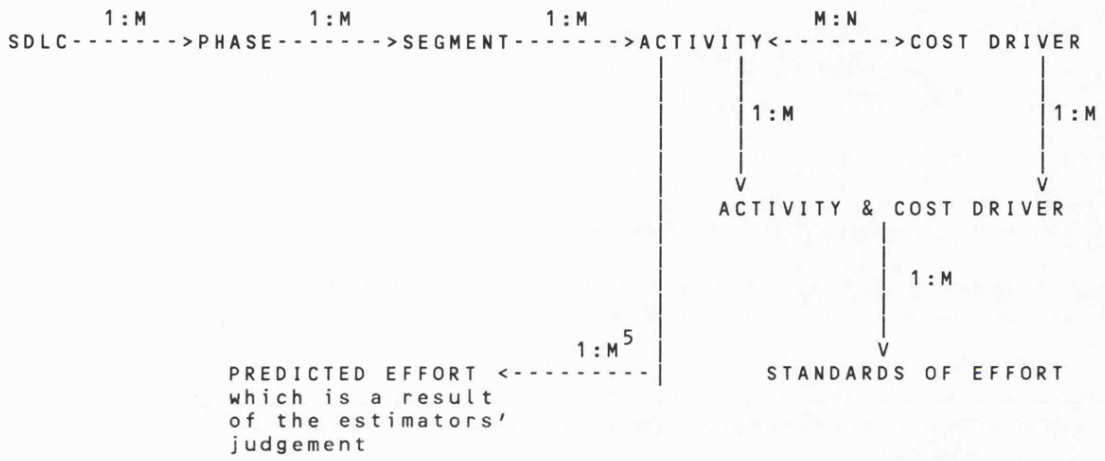


Figure 7.6 A partial view (c) of the conceptual data-model used in the EEM (A base model view)

7.5 COMPLEXITY AND RISK

Identification of the potential areas of complexity of a new system is a key factor in the effort estimating process, since the complexity is a prime cause for risk in software development. Risk is the traditional manner of expressing uncertainty and difficulty in the system's life cycle. The productivity rate is a function of the system complexity and difficulty and, hence, the level of complexity affects the effort and the duration of the project development. Therefore, the set of

---

.5. This is a 1:M relation, since resulting from projects histories, more than one prediction are included. The information included in the base model is an indication that this activity requires expert judgement for provision of the predicted effort and 'pointers' to projects where similar activities were implemented are provided.

'standards of effort' associated with a project is determined by complexity rules which present the 'general complexity' anticipated for the system as a whole, as shown in Figure 7.7.

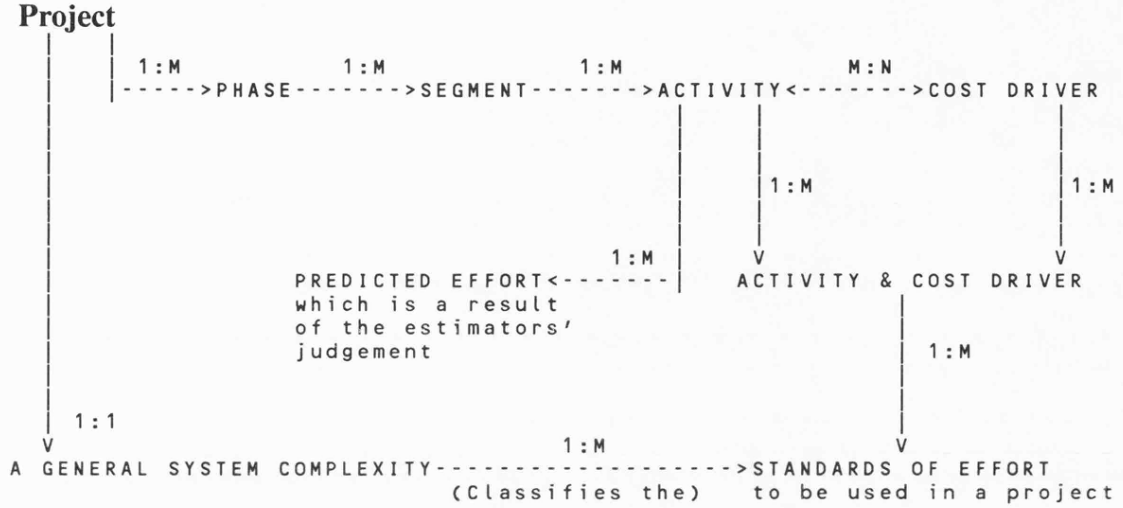


Figure 7.7 A partial view (d) of the conceptual data-model used in the EEM (A project view)

Examples are given to clarify how the concept described is implemented by the EEM in **Appendix 7C**.

In a quantitative sense, risk is the probability at a given point in a system's life cycle that predicted goals cannot be achieved given the available resources. Due to the complexity of risk components and the compounding uncertainty associated with future sources of risk, the thesis of this research is that risk cannot be treated with mathematical rigor during the early life cycle phases.[6] Uncertainty, complexity and risk are associated with the process of developing a software product. Their rigorous quantification is meaningless early in the life cycle.

---

6. As a system progresses through the life cycle and uncertainty diminishes, the degree of mathematical precision increases and can be used for the various measurements of complexity associated with the programming effort.

The complexity level of a system is a subjective classification and is based upon the difficulty of a project as perceived by the various parties involved who may differ in their assessments. They will obviously benefit from a tool which helps them in preparing themselves for the negotiation and the decision processes associated with the economic evaluation of a potential project.

The complexity metrics identified in this thesis are *a priori* metrics which aim to:

- Identify the appropriate 'standard of effort' values to be used by the EEM in estimating the effort required for the development of a software product.
- Evaluate the complexity associated with the development of the system and the risk of project failure (overrun of schedule or budget).

Complexity is often a problem of understanding. Understanding in this context is a function of the structure of the system and its size. Although large systems may be of linear structure (with limited decision points), they are assumed to be more difficult to understand and comprehend as result of the variety and connectivity they require. The knowledge about the problem at the outset of project development is not sufficient to allow a thorough assessment of complexity which stems from logical and problem oriented issues. Therefore, the assessments of complexity and the risk indicators in the EEM is based on data about the size of the product and about the complexity of the environments (system characteristics), as follows:

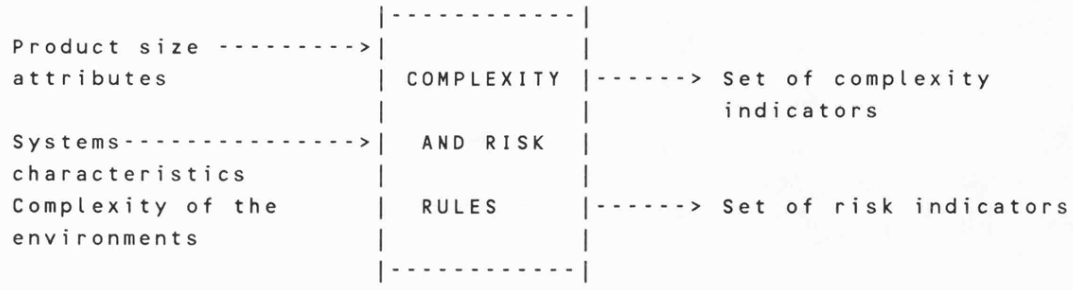


Figure 7.8 Complexity and risk

The level of complexity and risk involved in a system development is affected by various logical and problem oriented issues plus external factors, such as the organisational and technical environments and the project team composition. Thus, the EEM uses the following set of alternative indicators of complexity :[7]

* General system complexity.
* Organisational environment.
* Technical environment.
* Project team composition.

Appendix 7D includes the set of rules used by the EEM to assess each of these indicators.

7.5.1 Complexity and risk assessment

The assessment of the 'general system complexity' considers two groups of complexity determinants, they are attributes of product size measures (quantitative attributes) and environmental systems characteristics, as follow:

Group A: Attribute of product size

* Number of data elements in data-base.
* Number of logical data-bases.
* Number of complex/major functions, from user design viewpoint.
* Number of inquiry screens.
* Number of major reports in the target system.
* Number of reports in the target system.

---

7. This is based on the analysis in Section 5.8, the author's own experience and a cluster analysis on the data collected.

* Number of batch inputs (types, not volumes) or input documents.
* Number of on-line inputs (types, not volumes) for updating.

Group B: Environmental complexity determinants

* Required (realistic) response time for high volume transactions.
* System impact on financial status.
* System impact on operational status.
* Interface with other applications.
* Percentage of replacement of existing functions.
* User familiarity with the system.

In addition, when **evaluation of application packages** is assumed as part of the project, then the following complexity determinants should be considered:

* How many software packages, that were found suitable for in-depth evaluation, will be actually evaluated at the Preliminary System Design Phase?
* Level of customisation required. Number of modifications needed to fit the requirements.

The **'organisational environment'** considers the size of the company administrative and organisational structures, the pace and general systems effectiveness of the user organisation as external factors affecting the estimates. This is based on the assumption that an organisation with large and multiple committees will require significantly more project time than a small organisation with managers who are familiar with the system and the surrounding environments. The size of the organisation may be a major factor, due to the necessity of considering many viewpoints and of meeting the needs of diverse interests. The assessment of the **'organisational environment'** indicator is based on the following complexity determinants:

* Number of departments (other then the IS) involved with the project.
* Number of working units involved in the project. Is this project being developed for the usage of one working unit?
* What is the severity of procedural changes in the user department caused by the proposed system?
* Is this project a conversion or a functional repeat of a well known project?[8]
* Does the user organisation have to change structurally to meet requirements of the new system?

The following factors should affect the schedule of user oriented segments such as user requirements, user design, but mainly the system test.

* What is the general attitude of user?
* Number of people whose working practice will be affected by the system.
* Number of people whose working practice will be affected by the system, in one working unit. If great variation in unit size, then give an average.
* Decision makers.
* Information processing service structure.
* Commitment of the upper level user's management to the system?

The 'Technical environment' reflects the relative complexity of the information processing environment. It should be assessed in addition to the 'general system complexity' since the complexity of the application and that of the technical environment can differ. The following complexity determinants should

---

8. This question was phrased as an substitute to the question: 'percentage of replacement of existing functions', which is considered as an 'environmental complexity' determinants, in group B above. It was assumed that the response to both questions would be close. However, the results of the cluster analysis did not place them in the same group.

affect mainly technical oriented segments such as technical design and programming.

* Communication and distributed systems.
* Computer type, operating system, installation aids and project familiarity.
* Is any hardware new to the company?
* Data-base management system.
* On line monitor.
* Data dictionary.
* The development methodology.
* System architecture.[9]

The **'Project team composition'** reflects the experience of the individual project team members at the time of the Preliminary System Design initiation. The adjustment applied should represent a specific assessment of appropriateness between required and available skills. The estimates should be revised accordingly.

* Project team structure.
* Experience with industry/application.
* Technical experience.
* Staffing and Hiring considerations.

Although the EEM evaluates the complexity for each of the various categories, the results are not automatically incorporated in the direct calculation of the estimated effort for the PSD. The set of 'standard of effort' which is used in the estimation algorithm is classified only by the 'general system complexity', assuming a standard level of complexity for a project. The 'technical and the

---

9. This complexity determiner is currently included in the complexity assessment, however, it did not add any information in the sample data.

project team complexity' affect particular segments, mainly in the Construction phase. The 'organisational environment' impact is on the project risk assessment. Nevertheless, where a standard level of complexity is not applicable for a project, an alternate level of complexity may be assigned to a particular segment and or activity as further discussed in Section 7.8.1.

Risk and complexity assessments of a project are closely related processes, since both complex software products and complex environments are potential risk triggers. Thus, the risk indicators are a subset of the complexity determinants. They are:

* System impact on financial status?
* System impact on operational status?
* Number of departments (other then the IS) involved with the project?
* If you propose to replace the system, what percentage of existing functions are replaced on a one-to-one basis?
* What is the severity of procedural changes in the user department caused by the proposed system?
* Does user organisation have to change structurally to meet the requirements of the new system?
* What is the severity of procedural changes in the user department caused by the proposed system?
* Does user organisation have to change structurally to meet requirements of the new system?
* What is the general attitude of user?
* Staffing and Hiring consideration.

Yet, there are unique risk indicators which are not complexity determinants. They aim to account for schedule risk associated with a project. Since the project schedule is heavily dependent on the estimated effort, the following two risk determinants should be added:

* Total effort in PM.
* Project duration in months.

The set of rules used by the EEM to assess the risk indicator, are given in **Appendix 7D**, Section 7D.6.

The concept described in Section 7.5 is shown in Figure 7.9.

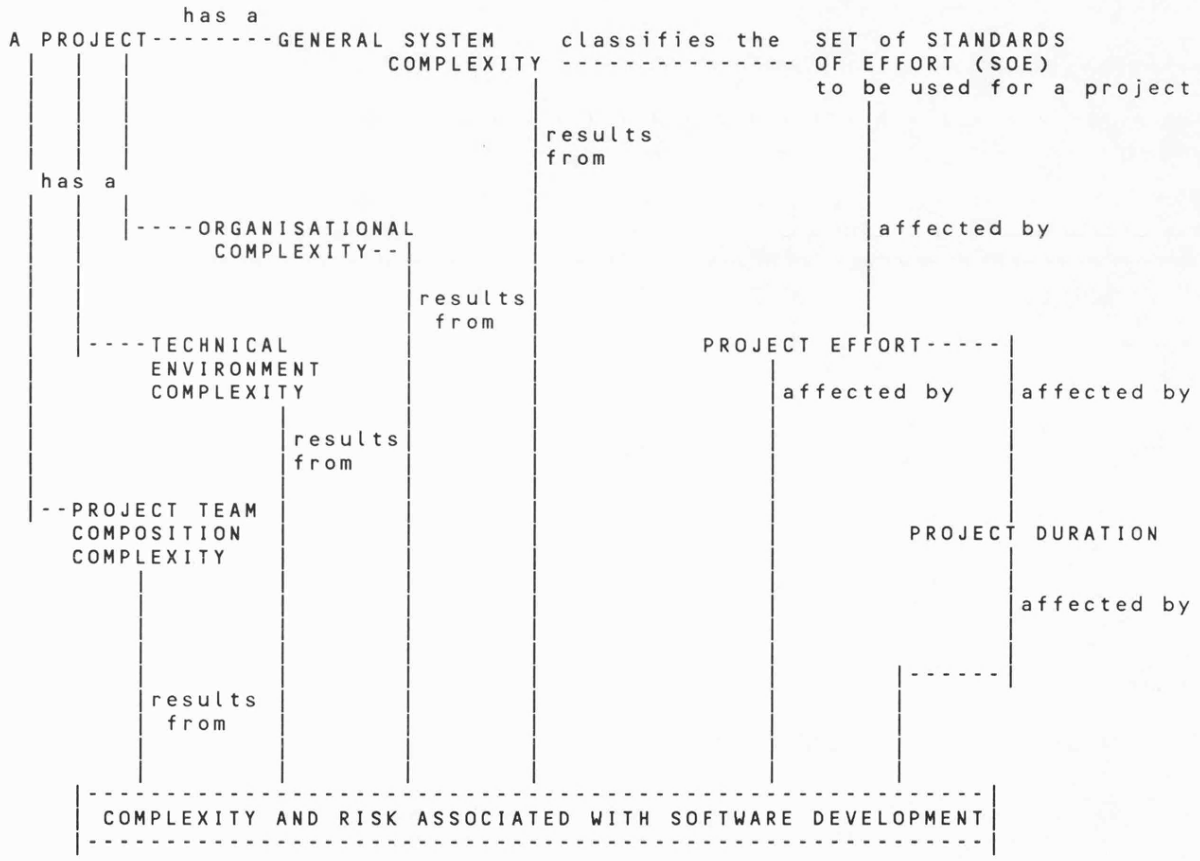


Figure 7.9 Complexity and risk assessment

7.6 WHO IS THE ESTIMATOR?

The estimation process can be implemented by an individual estimator or a group. The group should be composed of the project team members and/or various data processing personnel, experts from the user's department and/or external consultants who use their best judgement in order to determine the different aspects of the project profile. A group can work in one of the following ways:

- Each group member operates the model separately and eventually the model proposes estimates based on the data provided by all members. In this way the members act individually and the model computes an average of the group estimations based on the weighted answers of the team members.[10]
- The group uses the Delphi or some other group decision process in order to achieve consensus among the members. One agreed answer is input into the model. This process might be important for the assessment of the cost driver, which are the size attribute of the project.

7.7 THE EEM'S ESTIMATION PROCESS: ALGORITHM, ITERATION AND JUDGEMENT

The software estimation is an interactive and iterative process. The model uses different processes for the production of estimates. It works in cycles as discussed in the following paragraphs.

A schematic description of the processes encompassed by the EEM is given in Figure 7.10.

---

10. An attempt was made to place greater faith in estimates given by experienced project leaders/analysts (measured in years of experience). However, this attempt did not gain support in the walkthrough sessions. Most of the projects' data collected in this research were estimated by individuals

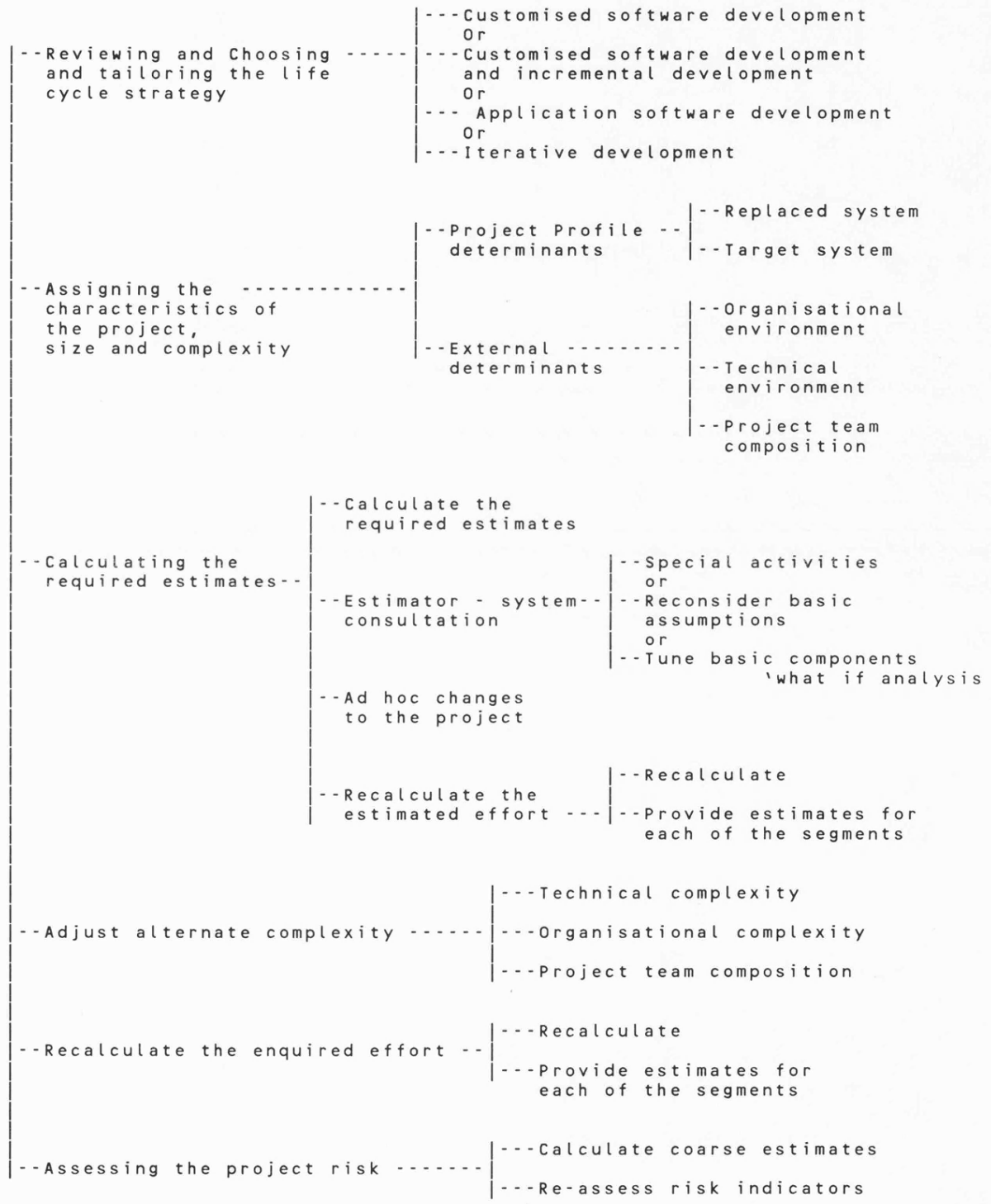


Figure 7.10 The EEM's function chart.

7.7.1 The first cycle: Reviewing, choosing and tailoring the SDLC strategy

Based on strategy decisions already taken, the estimator is asked to indicate which development strategy is to be followed, for example, whether software packages or an incremental approach are to be used. The estimator is then presented with the segments and activities which comprise the Preliminary System Design phase.[11] He is asked to indicate which of these segments and/or activities will be incorporated in the particular development process. The estimator is authorised to delete segments and/or activities not being implemented. Similarly, he can either reduce or increase the effort required for a segment or an activity. The effort required from data processing sources may decrease for some segments or activities, however, it may increase for others when external entities take part in the development process. External entities are, for example, the user organisation or consultancy firms. The effort estimated for co-ordination should be increased while the effort estimated for the particular segment or activity which is implemented by the external entity may be reduced. The assumptions and their reasoning are recorded and will support the calculation of the estimated effort.

Consider the following example. It is decided that the 'user requirements' will be identified, defined and specified thoroughly by individuals from the user organisation. This decision may reflect the estimates in several scenarios:

Scenario A: The effort required by the contractor organisation for the development of the particular project should be reduced.[12] Such a decision affects mainly the 'user requirements' segment which includes the following activities:

---

11. See **Appendix 7A** and **7A1**.

12. A contractor organisation may be the data processing department or an external software house.

* Review present system
* Identify functional requirements and information needs
* Identify performance and control requirements

However, it should be clear whether the user's organisation will be able to perform the whole segment without any help from the contractor organisation.[13] In addition, the contractor may decide that he is willing to accompany the process with one of his analysts, in order to permit continuation of the process. Hence, this decision regarding the development process has an impact on the estimator's assumptions. The estimator has to decide how to incorporate these decisions and assumptions into his estimates. He may do it in several ways:

- **Alternative 1,** the estimator assumes that the contractor organisation will need only 10% of the total effort estimated for this segment.
- **Alternative 2,** the estimator assumes that the proportion of the effort which will be implemented by the contractor in this segment will be, for example, as follows:

*	Review present system	5%
*	Identify functional requirement and information needs	10%
*	Identify performance and control requirements	15%

Taking different assumptions is likely to result in different efforts. The different assumptions might be used in different estimation sessions which are targeted on different objectives. However, the second alternative allows the EEM to produce a **detailed work plan** which is based on activities.

Scenario B: Although the decision was that the 'user requirements' (external

---

13. The decision maker might not be aware that technical difficulties might be encountered in implementing the last activity.

requirements) will only be identified, defined and specified by the users' organisation, the effect on the estimated effort for the 'user design' segment should be also considered. Since the 'user design' segment is usually best performed by a team of users, system analysts who are familiar with the business domain and technical analysts, they should be involved in the implementation of this segment. It includes the following activities:

* Define inputs and outputs
* Define processing functions
* Define data requirements
* Issue preliminary functional and technical specifications

The estimator should clarify whether the user's organisation aims to be involved in the implementation of this segment and if so to what degree.[14] Based on his judgement, the estimator may conclude with the following example of allocation of effort for the contractor organisation:

* Define inputs and outputs	60%
* Define processing functions	100%
* Define data requirements	80%
* Issue preliminary functional and technical specifications	10%

Scenario C: The estimated effort for organising the project should be increased. Since the number of team members is considered as the cost driver for 'organising the project'. It should be (either proportionally, or based on the estimator judgement) increased according to the number of participants, either

---

14. The decision makers may not be aware that the activity 'define processing functions' includes the functions which support the flow of information among work stations. This activity cannot easily be implemented by the user staff. Ideally, when such a decision is recorded, the EEM should direct the estimator to explore the exact meaning of the decision and its implications. This is included in the agenda for further development of the EEM.

full-time or part-time.

Scenario D: The complexity associated with the development may be reduced. This stems from 'user involvement' being a factor which reduces complexity and improves productivity or, more correctly, effectiveness. This will be present in the complexity assessment.

The appropriate life cycle route for the development of the particular project has been chosen and tailored to the strategy decisions taken. The estimator's assumptions and decisions were incorporated into the life cycle approach and recorded. Moving to the next session in this cycle: the profile of the project is assessed.

7.7.2 The first cycle: Assessing the characteristics of the project

Since a prime critical success factor in software development is the prevention of uncontrolled risk, one of the first questions the parties have to ask themselves is how risk-laden the project is from the managerial and technical points of view. Hence, complexity and risk analysis should be conducted. The EEM initiates a series of questions to which the estimator is asked to respond. The evaluation of the project characteristics such as size, complexity and risk will be based on the estimators response, and the corresponding rules which reside in the knowledge-base. Two main data entry sections are included in this cycle. These serve to record the estimators' assumptions. The first section records assumptions regarding the **project size** and the second the **complexity** and the **risk** associated with the projects environments.

Attributes associated with three different sources: the target system, the replaced system and the process of implementing the PSD, are used for sizing the development project, e.g.: number of screens, reports and inquiries in the replaced system and in the target system, number of data elements, file conversions and re-designed forms needed to support the functionality of the target system, and number of tenders to be evaluated, number of high volume transactions, for which capacity planning is needed. The answers given by the

estimator are based on the general knowledge available in the organisation, as a result of the 'preliminary survey' (in the Project Planning phase).

Although the EEM evaluates the complexity for each of the various categories, the results are not incorporated in the direct calculation of the estimated effort for the PSD. The set of 'standard of effort' values which is used in the estimation algorithm is classified only by the 'general system complexity'. The reason is that some attributes associated with the organisational environment are considered as **project size** attributes (e.g. number of interviews, number of tenders to evaluate, contracts to negotiate). The technical complexity and the project team complexity affect particular segments, mainly although not explicitly, in the Construction phase. They should considered separately and at present it is subject to estimator judgement.

7.7.3 The second cycle: Consultation session, estimator - EEM

The second cycle includes the following:

* Calculating the first iteration of the required estimates.
* Recalculating the required estimates following a consultation: Estimator - EEM.

At the beginning of this cycle, the model tries to calculate the estimates. It may encounter difficulties in doing so with regard to a few activities characterised by a high variability of effort needed to accomplish them in different projects. For example, the 'hardware and software selection' process will vary among organisations and types of projects. We cannot compare this process when done for a governmental agency with that for a company in the private sector. Similarly, we cannot compare the establishing of selection criteria for a data-base management system (DBMS) software to be installed in a mainframe computer that supports a distributed system, with that of selecting DBMS software for a micro computer which aims to support software

development in user departments.

Wherever the model comes across activity of that sort, it consults the user and uses it to consequently generate an estimate. If the estimator feels that the estimates do not accurately reflect his opinion, experience, intuition, etc., he informs the model, which responds by presenting him with all the questions and the answers, for his reconsideration. The estimator is allowed to change them and the model will provide new estimates.

The **end product** of this cycle is an estimate for the number of working hours required to accomplish the activities included in each of the segments which compose the Preliminary System Design phase. The required number of working days is derived from the estimator's answers, the knowledge-base of the model and the inference rules which manipulate both. The EEM will also inform the estimator of the major assumptions that affect the estimates. The algorithm for calculating of the estimates is given in Section 7.8

7.7.4 The third cycle: Fine tuning the EEM to the specific environment

The third cycle includes the following:

* Examining the principal components of the knowledgebase and making ad-hoc changes to them.
* Recalculating the required estimates.

The prime concern of this cycle lies with fine tuning the major components of the model to the specific organisation or project.

If at this time the estimator is still not satisfied with the estimates, the principal components of the knowledge-base are shown for his examination in the following two iterations:

- The first iteration covers the rules used by the model in determining the project complexity level. The estimator is asked to examine the rules

included in the knowledge-base for the determination of the complexity level. He is authorised to change the complexity level on which the estimates are calculated if, in his opinion, it is not suitable for this project environment.

- The second iteration covers the cost drivers which contribute to each of the activities and their correlated 'standard unit of effort'. It is suggested that the estimator examine the:

  * Associations, proposed by the model, between the cost drivers and the activities.
  * The 'standard of effort' (SOE) units attached to each of the integrated entities composed of cost factor and activity.

These components are also subject to an ad-hoc change by the estimator. The EEM offers the estimator a 'what if' analysis option for that purpose.

The cycle culminates by recalculating the estimates for the effort required for the PSD phase.

7.7.5 The fourth cycle: Providing a coarse estimate for the total project effort and re-assessing project risk

The coarse estimate for the total effort can be obtained now. This is based on the estimator's judgement about the resource distribution among the phases of life cycle in the particular development process. The EEM may suggest guidance for the classification of the project based on size, development strategy, and environments.

Based on the coarse estimates for the total effort the risk indicator should be re-assessed, since target effort and duration required for its implementation are themselves possible sources of project risk.

7.8 CALCULATION OF THE ESTIMATES

Having described the conceptual framework of the model, the software development effort for the Preliminary System Design phase is calculated, as follows:

1. The **life cycle** components take the form of a binary M by N matrix *(L)* in which each cell has the value of either 1 or 0, respectively indicating that the identified activity does or does not have a cost driver that contributes to it.

   Lij = 1 means that cost driver j does contribute to the cost of activity i

   While,

   Lij = 0 means it does not.

   Hence, the dimensions of the matrix *Lij* are:

   * Activities i, i = 1....M
   * Cost drivers j, j = 1....N

2. The assessment of the **project profile** results in values of size attributes, which represent the work needed for the phase, being estimated. The size of the project is defined by a vector *dj*, where *dj* is the number of times cost driver j is used.
3. In order to estimate the effort, the life cycle matrix *Lij* must be related to the number of times each cost driver is used. This is achieved by calculating matrix *Dij,* showing the units of work required for activity i and all the occurrences of cost drivers in the overall project. Therefore,

   (7.1) $Dij = Lij * dj$

4. Matrix *Kij* includes the **'standard of effort'** of activity i and cost driver j. There are **three levels of complexity** (c = 1,2,3) corresponding to simple, moderate and complex. There are three matrices $K^c ij$ which store the 'standard of effort' for each complexity c, activity i and cost driver j. The content of each matrix will be the 'standard of effort' units (person-hours) with each of the associated activities and cost drivers.
5. To calculate the effort for a project with complexity c, we need to calculate $E^c ij$. This is achieved by multiplying the following matrices:

$$(7.2) \qquad E^c ij = Dij * K^c ij$$

This assumes that there is no adjustment for software packages.

6. In order to estimate the effort for a project incorporating the search for application software packages and their usage, there is a need to employ an adjustment factor *Sij* which reflects the reduction of effort required to accomplish activity i and cost driver j. Each cell *Sij* includes an effort adjustment factor, associated with activity i and cost driver j. The estimated effort will be achieved through the multiplication of the following matrices:

$$(7.3) \qquad Es^c ij = E^c ij * Sij$$

Where,

Es indicates the estimated effort assuming an adjustment factor for application software.

7.8.1 Assigning alternate complexity

Using the above procedure, the effort has been estimated assuming a standard level of complexity for a project. However, this is not always valid. No doubt there are projects which incorporate different levels of complexity for various activities or segments, to which we would like to assign an alternate rating of

complexity. These issues can be dealt with in either or both of the following ways:

1. Using a particular complexity factor which is correlated with a specific activity and cost driver. This will imply the replacement of **individual** 'standard of effort' cells (in the $K^{c}ij$ matrix) which had been found to be generally suitable for the project's level of complexity. Furthermore, a requirement to account for the specific categories of complexity might be considered. Segments and/or sub-segments being mostly affected by each of complexity attributes (categories) could be identified.
2. Adjusting a specific complexity factor to segments and/or sub-segments which are expected to be affected by the specific complexity attribute. This approach implies that, as a result of assessment of the complexity of the external environments, an adjustment factor will be applied to relevant segments.

Unfortunately, adopting the discrete comparison (standard approach) does not help in all cases. The variance between projects for the same activity may be of such an order of magnitude that no comparison will apply. The model suggests that the adjustment should be done by the estimator, using his expert judgement.

7.9 THE PRODUCTS OF THE MODEL

Based on the answers given by users of the system, the EEM knowledge-base and the rules for manipulating them, the model offers estimates for the manpower needed for each segment in the Preliminary System Design phase. From these estimates, the coarse estimates for the total effort can be calculated using the resource allocation which is the most applicable to the particular project profile and its environments. Having in front of him the detailed assumptions and decisions taken throughout the Project Planning phase and the estimation

session, and using his judgement, the estimator can decide on a different allocation of resources for the calculation of the coarse estimates. Project risk is re-assessed by incorporating these results.

7.10 A CASE STUDY - PROJECT 'A'

A case study is introduced here to demonstrate the main features incorporated in the EEM and the support it provides throughout an estimation session. The major outputs of the EEM will be introduced throughout this discussion.

The case study describes an enhancement project aimed to add and to update existing functions in a large data-base, using the ADABAS Data-base Management System. The project was developed, by an in-house data processing unit in the private sector, for a company in the banking, insurance and financial services segment.

The proposal for a software project is a product of the Project Planning phase. It includes decisions regarding the development strategy such as the use of tools and the level of involvement expected from the various organisational entities throughout the development process. The Project Planning Document (PPD) for the case study is given.

THE PROJECT PLANNING DOCUMENT (PPD) - PROJECT 'A'

The **Project Objectives** were to produce a computer system, which would provide a significant reduction in the amount of clerical work required to process scheduled 'ordinary branch' maturities and 'income bonds'. It was also to improve customer servicing through the improved design and quality of all associated forms and documentation.

The scope of the project covered the following areas:

- the processing of scheduled 'ordinary branch' maturities and 'income

bonds' automatically generated from the 'ordinary branch' update system.
- the clerical functions of the life claims department associated with this processing.
- the link between maturities processing, the Cashiers and Cheques Systems and the 'ordinary branch' Loans System.
- the introduction of newly designed claim documentation to improve presentation to the customer.
- the consideration of alternative methods for the printing of all maturity forms and documentation.
- the identification and addition of interest on delayed claims and associated tax documentation.
- the effects of any changes on the clerical functions of the House Purchase Public, House Purchase staff and Banking Service departments.
- the conversion of policies from the current to the new maturities system.
- the flexibility of the system to be extended to cover other claim areas in the future.

Costs and Justification. The volume of 'ordinary branch' maturities would increase significantly during the coming year and for several years thereafter. A full cost/benefit analysis is not shown here as this was produced as part of the System Proposal.

Project Organisation was established. The user manger, the Information System project manager and the project team members were identified. It also indicated that additional staff were to be added to the team (from department X) as the amount of development work increased. Additional Life Claims staff were required during the testing and the training phases of the project.

The reporting links and responsibilities were established. The individuals who were in control of the user team, the IS team, the liaison officer and the training were identified.

Key assumptions. Experienced Life Claims staff, in addition to project team members, were to be available to carry out testing of the system and supervise training during the period leading up to implementation.

Resources were to be available from other departments to carry out interface testing within the time scale of the project.

The re-design of forms and documentation, the printing of stationery and the testing of it were carried out before implementation.

System development approach was based on the use of Natural 2 for the online applications. Batch applications were to be written in Cobol or Natural 2 as appropriate.

7.10.1 An estimation session, using the EEM

Each EEM estimation session started with a life cycle editing session. The estimator analysed the life cycle strategies, choosing the appropriate strategy and modifying it based on the strategy decision taken.

Life cycle editing and recording strategy assumptions

At the start of an estimation session, the EEM presented the life cycle strategies from which the estimator chose the relevant strategy for his development task. The estimator was now referred to the segments encompassed in the chosen development strategy to which he incorporated the strategy decisions which were included in the PPD, as follows:

- The **'organisation segment'.** Since the number of team members (part time or full time) is the major cost driver for organising the project, the effort for the 'organisation segment' was increased to account for the user involvement. In project 'A' the contractor's organisation was to use 3 part time staff and the user planned to involve one part timer in the PSD work. Therefore, increasing the effort required for this segment by a third was considered an appropriate compensation for the extra user involvement. The estimator recorded this assumption by indicating the proportion of

the total effort to be used for the 'organisation segment' as 1.3. (The estimator could have used a different ratio, however, in this case study this assumption was agreed upon). He was then asked to record the reason for adjusting this segment.

- The **'user requirements'** and **'user design'** segments were to be implemented mostly by the users. It was assumed that the users would produce half of the effort required for this segment.
- The **'technical design'** included the design of the technical architecture, data-base and system processes. Since a data-base was largely, already, in existence, only 3 new logical files and references to nine existing files were planned. The effort required for this segment was reduced accordingly.
- The **'installation schedule'** segment was reduced to a tenth of the level of effort expected for this segment as no new hardware and software was part of the project.

Examples screens for this process are presented in **Appendix 7E.**

Inputting projects data

The session continued with an assessment of the project profile. The EEM initiated a series of questions to which the estimator responded. The session included inputting the values for the size and the complexity attributes. Whenever the EEM came across values corresponding to the tasks characterised by high variability of effort required to implement them in different environments, the EEM initiated a request for additional information. In the case of project 'A', the EEM came across values for size attributes which were considered to belong to the categories of concern. The following groups were identified:

- Control and utility modules. For example, modules to manage file access and track updates, produce control reports, data-base management

control routines.

- Complex or major functions from the user design viewpoint. They were based on the business functions.
- Complex process from the technical design viewpoint, which were not directly related to input or output.

The EEM requested the additional information and received the following:[15]

- 180 hours were planned for the accomplishment of 20 identified control and utility modules. These modules were concerned with the restriction of access to parts of the online system and dialogue switching via the use of tables.
- 60 hours were estimated as being required for the accomplishment of an allocation strategy to work in totally flexible way and of the design for total user control of batch running of the system.
- 100 hours were estimated for the accomplishment of a procedure to convert numbers (money terms) to words etc.

Assessing the profile and the complexity of the project and its environments

The values for the size and the complexity attributes were updated. The data was analysed and the session culminated with the complexity assessment. The 'general system complexity' which determined the 'standard of effort' (SOE) to be used was presented. The estimator was asked to approve the EEM's recommendation or otherwise change it. To support this decision, the EEM provided the results for each of the categories of complexity and risk. This presentation was followed by detailed information on the contribution of each

---

15. These estimates were, as anticipated, for the total effort the PSD and the Construction phases.

component which was incorporated in the calculation of the complexity and the risk.

In the case study, the EEM recommended that the **'general system'** was complex, however, at the lower end of the range (42 points, in the range 41 to 70 points for a complex assessment). The risk associated with this development was considered as moderate (16, in the ranges of 8 to 23 for moderate risk). The major contributors to this were the requirements for a high response time and the multi-directional interfaces with several applications involved in this project. However, the project was a functional repeat of a well known project and the users were planned to participate actively in the development process as an integrated part of the project team. The additional categories of complexity were all assessed as moderate.

The **'technical environment complexity'** was anticipated to be moderate (18 in the range of 7-20). Although the system incorporated extensive communications, the communication facilities system was well established and there was no need for modifications. The hardware environment was complex but the project team was familiar with it. The **'organisational complexity'**, was anticipated as moderate (22 in the range of 10-29) and so was the **'project team complexity'** (8 in the range of 4-11). The team was very small with a single decision maker. All staff required were available and familiar with the system, the team had considerable previous exposure with industry, yet had limited knowledge with the specific area of application, therefore a small learning curve was anticipated. The determinants contributing to each category of complexity were presented.

Analysing this information the estimator could safely decide on a moderate level of complexity for the estimates calculation. He confirmed his decision and the EEM chose the moderate matrix (K^2ij) of SOE to be used in the algorithm for the effort estimation, as defined previously in Section 7.8.

ADABAS was the main DBMS used by project 'A'. The DBMS and NATURAL/2 was mainly used for screen formatting. The impact of the use of these tools was considered. Since firm data on this issue was not available when building the model, a decision was made to use (as a starting point) the SOE

matrix *(Sij)* which is used by the EEM when the application software strategy is considered. The question was asked whether the same level of effort and distribution of resources (between the PSD and the Construction phase) could be assumed, at that macro level of coarse estimates, when using ADABAS and NATURAL/2 and when assuming the application software approach. The similarity between the use of a screen formatter, report generator in such applications and the use of software packages in general systems development was considered to be sufficient to attempt this combination.[16]

Consultation session, estimator - EEM

One of the original and important features of the EEM is that it estimates all the effort required for the PSD development process. This stemmed from the main objective for building the EEM, namely the wish to provide support for all parties involved in the management of the software development. High level decision makers needed to know the total organisational effort and not only the effort consumed by the contractor organisation. The EEM was therefore built to support these decision making situations. The estimator was able to evaluate his own part in the development process given that he knew the proportion of his contribution to the effort.

All the data required was now available to the system. The EEM was able to analyse the data provided by the estimator, by using the knowledge-base and the inference rules and to propose estimates for the PSD effort. Three curves presenting the minimal, the most likely and maximum estimated effort for each segment in project 'A' were shown on the screen. The peak and the lowest estimated effort were indicated.

Estimated effort from the first attempt was presented. The output included the effort in person hours (PH) and the percentage for each segment in

---

16. This assumption was tested on 2 additional projects and found to be appropriate, however, additional research is required to confirm this.

the PSD. To allow a top-down comparison of the EEM estimates with the industry 'average', a 'common' distribution of effort for the two main strategies for software development was also presented. Assumptions and decisions taken at the life cycle editing session were shown, although they were not yet incorporated into the calculation of the effort. They were given for the estimator to re-evaluate.

Additionally, the contribution of each of the cost drivers to the total effort was given for the purpose of analysis. The preliminary estimates were presented for project 'A'.

These were preliminary results in which the decisions made at the life cycle session were not yet embodied in the calculation. The estimator realised that the estimated effort for the 'technical support' segment was high in comparison with the industrial average distribution of effort. The complex processes and complex functions contributed to this segment. The estimator might have been willing to reconsider his assumptions about the effort embodied in these processes. Or he might have considered approaching the users regarding their requirements (design-to-cost procedure). The high effort for the 'technical support' was an indicator which should have directed the attention of the estimator, the user and the software developer to the existence of issues which may require special solutions.

In addition, the estimator of project 'A' learned that the estimated effort for the planning of the file conversion effort was about a third of the total estimate for the PSD work. The other major contributors were the system overhead and the effort associated with the re-design of forms.

If the estimator of project 'A' had been the manager who had the overall responsibility for the development of the project, he would have learned that the effort accumulation would reach the peak when 470 PH were consumed.

The decisions taken thus far were taken into account and were presented in the subsequent screen. The estimator was still given the opportunity to analyse and change some of his assumptions, or to try the effect of additional assumptions. He could have added, subtracted or multiplied the effort for each of the segments. The changes were followed by requests to provide the reasons

for the alterations made. The changes and reason for the adjustments were recorded.

A decision was made (independently of the estimating process) to postpone the screen formatting effort to the Construction Phase. The cost drivers contribution to the PSD effort showed that the effort associated with screens and enquiries which were required in the target system was estimated at 62 PH (for the users and the contractor organisation). Since the users were involved in this process, a decision was that only 40 hours would be reduced from the estimated effort on this account. Using the last output screen, 20 PH were subtracted from the user design and the from the technical support segments.

The last output screen in this series presented the effort calculated by the EEM incorporating all the decisions taken.

Additional information was available for the estimator to support the decisions taken throughout these estimation and the consultation sessions. The estimator may have wished to analyse the project profile based on his assumptions of the application domain and its environment.

Examples screens of this process are shown in **Appendix 7E.**

7.10.2 Analysis of results

The actual effort required for the PSD was 490 PH, for Information System's personnel only.[17]

The EEM calculated the effort, on moderate complexity, as 530 person hours which is 108% of the actual effort required. The estimator's view of the project estimated the effort as 420 PH which is only 85.7% of the actual effort. Based on the assumption that the resource allocation between the PSD Phase and the Construction Phase is around 17%, the total effort required for the project development is extrapolated.[18] The coarse estimates for the the total

---

17. This effort did not include the user who was assigned to the project.

development effort for this project are 3118 hours, which is 106% of the actual effort for this project (2930 hours). The project team estimated the effort required for the total project to be 3150 hours which account for 107% of the actual effort. A summary is given in Table 7.1.

	PROJECT DATA		EEM	COMPARISON (%)	
PHASE	Actual effort	Team estimates		Estimates vs. Actual (Actual effort = 100%) project team	EEM
Preliminary System Design	490	420	530	- 14.3	+ 8
Total project	2930	3150	3118	+ 7	+ 6

Table 7.1 Actual versus estimated effort - project 'A'

7.10.3 Conclusions - Project 'A'

The EEM estimates for the PSD phase are more accurate than the project team's estimates (-14% to + 8%), the EEM coarse estimates do not differ from the those estimated by the estimators of the project. Both are quite accurate.

However, the quantitative analysis is only one side of the coin. The process demonstrated using the project 'A' case study demonstrated the features embodied in the EEM and the ability to support the management of software development, emphasising the effort estimation process.

The flexibility of the EEM to cope with changing situations was demonstrated in the situation where a decision was made to move the activities

---

18. These coarse estimates are based on the assumption that resource distribution among the phases of software development, is approximately 20% for projects which are characterised as belonging to the Organic Mode [Boe81]. However, prototype requirements, using screen formatting activities were postponed to the Construction phase (to the detailed Design segment) therefore, the assumption to account for 17% only, is considered as appropriate. This agrees with the actual resource allocation for this project.

associated with prototyping of the screens to a different stage. The usefulness of the outputs in supporting various parties was also shown. Similarly, the EEM handles quite easily a change in the assumption to use the 'application packages factor'. The only thing for the estimator to do was to indicate this to the EEM whilst repeating the analysis stage.

7.11 A CASE STUDY - PROJECT 'B'

Additional case study is introduced with the aim to analyse various characteristics of software development, their impact on the effort required for the development and the way the EEM deals with it.

The **project's background.** Project 'B' was a development of entirely new system which aimed to support a set of processes previously implemented manually, in multi-sites, scattered all over the country. The system was developed by an in house IT department in the public sector. The project involved the selection of hardware and software, and interfaces among various types of hardware.

Project 'B' was developed using a report generation language and an application generator, both in house developments. Whilst the report generator was a long time in place and usage, the application generator was a new development, project 'B' being the first software development to use it.[19] The application generator evolved and under-went changes during the development of project 'B' and as result of its use in the project. The evaluation of the application generator was a part of the project mission, although informally. 90% of project 'B' was implemented using the application generator and the remaining 10% in Assembler. All the reports required produced using the report generator.

---

19. It might be worth explaining what is meant by application generator. An application generator is defined as *"a tool which produces and executable application from a non-procedural source language, but the application thus created uses services and features from the application generator and can only executed with the generator as a host* [Jon86].

All staff required for the development were available, yet a major learning curve was anticipated resulting from the type of the application, the use of entirely new hardware and new application generator.

Some **size attributes** of project 'B' are given below. The list of size attributes used in the EEM can be seen on the examples screen in **Appendix 7F.**

* Project size in LOC was 70K.
* The project involved about 50 interviews.
* The replaced manual system had about 50 reports and about 120 input forms.
* One site, for example, included more than 100 local work stations and around 20 remote work stations.

7.11.1 Recording life cycle assumptions

The following adjustments were made to the life cycle base model, with respect to the activities associated with project 'B':

- Only 50% of the activity 'define inputs and outputs' which is included in the **'user design'** segment were implemented by the IT group. The rest was implemented by the user.
- The **'technical support'** segment was not implemented, testing and conversion processes were not designed and the resource requirements for them were not evaluated. There were no good reasons for it.
- Only 20% of the activity 'develop conversion approach' which is included in the **'installation schedule'** segment was implemented.
- Contracts negotiation was not in the responsibility of the IT group. Therefore, the activity 'negotiate contracts' which is included in the **'hardware and system software selection'** segment was eliminated.
- In the **'Cost/benefit'** segment, the operating costs and benefits of the system were not fully evaluated. Thus, it was assumed that only 20% of

the required effort will be consumed.

7.11.2 Assessing the complexity of project 'B' and its environments

The **'general system complexity'** for project 'B' was assessed by the EEM as complex, (41, in the range of 41-70 for complex score) although on a very low level. The risk associated with the project was recommended by the EEM as moderate (16, in the range of 8-23 for medium risk).

The major contributors to the **'general system complexity'** were the following:

* Amount of reports, although most of them minor reports. The target system required 50 reports.
* Six complex function and three complex process were required.
* The data-base involved more than five logical data-bases.
* The required response time was high (1-3 seconds)
* The interface with other application was a multi-directional with several applications.
* The impact on operational status was critical.

The **'organisational system complexity'** was anticipated to be moderate, but at the highest level (29, in the range of 10-29 for moderate score). The contributors to this score were:

* More than three departments (other than the IS) were involved with the project.
* Although the project was developed on a single-site, it was intended for multiple working units within multiple-sites.
* The project was an entirely new system. The algorithm and logical design were developed from scratch.
* The decision process associated with the project was considered as

complex. Multiple committees were involved and multiple reviews were held.

The **'technical complexity'** was assessed as complex (26, in the range of 21-35 points for complex score). Although the system incorporated communications, the communication system was established but there was no need for substantial modifications. The reasons for this score were namely:

* All hardware was new to the entire project team.
* File management was customised. No DBMS was used.
* A new on-line monitor was used.
* Communication facilities were established but some development required support.
* The system architecture involved distributed facilities, however, a centralised data-base was used.

'Project team composition' was assessed as complex, although, at the very low end (13, in the range of 12-20 for complex score). The reasons for this score were the following:

* The team's experience with the application was minimal.
* A major learning curve was anticipated.

It was also the team's first exposure to the use of an application generator.

In **summary**, each of the complexity categories was considered as either complex, or very high on the moderate scale. This might indicate a need for adjustments of particular segments.

The impact of the use of the two generators in the development process should be also incorporated into the estimates, however, differently. **Application generators**, usually, are most effective in the internal design and coding segments, meaning in the Construction phase. Their impact on the requirements, user design and technical support and on the planning of testing and conversion

processes are minimal. The impact of their usage is minimal also on the user documentation and integration. In addition, in the case of project 'B' it was clear that the effort required for the development will not be reduced. On the other hand, it was clear that it should consume additional effort and duration, resulting from the evaluating process and the learning curve anticipated.

The use of a **report generator** impacts mainly the Construction phase, its impact on the user requirements is also minimal, however, the impact on the user design might be greater, although not significantly. The effect on the Construction phase is significant in the coding segments (a fall of about 80% in the effort might be observed), testing and error correction is easier when a report generator tool is used. The overall impact on the Construction phase might be as high as 40-45% of the effort.

7.11.3 The EEM estimation session and the outputs

The first piece of information the estimator received resulting from the EEM analysis of the data was the contribution of the various cost drivers to the total effort (the life cycle adjustment were not yet incorporated). The results are shown in Table 7.2.

The results indicated clearly the high effect of the concurrent evaluation of hardware and software on the development effort. Although the activities of tendering and evaluating the proposals took part mainly in the PSD, it had impact on the rest of the development. The effort required for producing the new reports and screens was also substantial. Capacity planning effort was associated with project 'B' and the EEM recommended an estimated effort of 2736 hours for this activity.

The effort consumption at the peak manning level was estimated as 4080 PH, and the minimum was 2040 PH.

Cost driver	Hours
System overhead	694
Project team members	100
No. of interviews	301
Old reports, screens and inputs	348
New reports	800
New screen, messages	1856
New inquiries	8
Re-designed forms	56
Data-base size	346
Invitations to RFP	1200
Evaluations of RFP	6460
High volume online inputs	2736
Utility and control	125
Special file conversion	175

Table 7.2 The contribution of the various cost drivers to the total effort

The **Basic estimates** were shown, including the distribution of the effort among the segments of the PSD. The impact of the evaluation of H&S was clearly indicated. The EEM estimated that these activities consume about 50% of the effort required for the development of the PSD, by all parties involved in the development effort. The total effort for the PSD was estimated as 15444 PH for all parties involved in the development, and when the changes to the life cycle were accounted the PSD effort was estimated as 12380 PH.

At that stage the EEM initiated a screen on which the alteration required can be made and their reasons recorded. In the case of project 'B' the estimator decided to incorporate two additional changes, as follows:

- The 'user requirement' segment was increased with 10% compensating for the anticipated difficulties in the development as a result from the unique

issues described previously.

- The 'user design' was reduced to 85% accounting for the use of the report generator. This reduction considered the effect on the Construction phase also.

This resulted in the EEM estimates of **12263 PH** for the **PSD.**

Examples of the screens supporting the EEM process are given in **Appendix 7F.**

7.11.4 Analysis of results

The EEM estimated the effort required for the **PSD** for IT personnel to be **12263** PH, which was **101%** of the actual effort required. The actual effort required for the **PSD** was **12096.** The project team did not estimate this part of the effort. The EEM estimated the effort required for the total development to be **40052** which was -12% of the actual effort. The actual effort required for project 'B' was 46368 PH. The project team estimated it to be 24192 PH. The EEM coarse estimates were based on the assumption that resource distribution between phases of software development is approximately 25%, for the Project planning and the Preliminary System Design phases, for projects of that type. The actual versus estimated effort is shown in Table 7.3.

	PROJECT DATA		EEM	COMPARISON (%)	
PHASE	Actual effort	Team estimates		Estimates vs. Actual (Actual effort = 100%) project team	EEM
Preliminary System Design	12096	***	12263	***	+ 1
Total project	45368	24192	40052	- 48	- 12

Table 7.3 Actual versus estimated effort - project 'B'

7.11.5 Conclusion - Project 'B'

The EEM estimates are significantly better than the project team estimates. One reason is that the EEM accounts for activities which are not usually considered such as tendering, evaluating of proposals and capacity planning. Those are time consuming activities which affect the effort and the duration of the total project. In addition, the EEM didn't compensate for the usage of application generator, while the project team did. They estimated that the use of application generator will reduce significantly the development time even though it was the first exposure of the project team to this strategy of development. It is worth noting that the in house application generator was not the only one considered for project 'B'. Additional software was evaluated for this purpose, however the project team concluded that the in house development match best their needs. The project team did not evaluate the difficulties anticipated with the users and those raised from the implementation of system that was never computerised but worked quite smoothly manually. Most of these factors are included in the complexity rules used in the EEM and thus classify the SOE for the specific development.

Chapter 8

EVALUATION OF THE EEM

8.1 INTRODUCTION

Two avenues of evaluation are considered in this chapter, a qualitative and a quantitative analysis of the EEM. The qualitative analysis includes a summary of informal discussions and a comparative evaluation. The comparative evaluation follows the process employed by the IIT survey of sizing models done for the US Air Force [IIT87a]. The IIT report concluded by rating each model according to its relevance to its intended usage. The criteria suggested in this report are of general relevance to effort estimation models. This method was used to compare the EEM with related approaches. All the scores for the EEM are based on the prototype currently in use.

The quantitative analysis uses two tests, they are: Conte's [Con86] evaluation criteria as discussed previously and a linear regression. The regression analysis is used as a means of measuring and comparing the relationships between the estimates yielded by the EEM and by the projects teams, and with the actual effort. A comparison of the EEM coarse estimates with the actual results and with the original estimates as produced by the project teams are then presented for a sample of 18 projects.

8.2 EVALUATING THE EEM - QUALITATIVE ANALYSIS

When evaluating a model, the motivation for its development and its objectives should be considered. Four long-term research goals were stated in the opening chapter of this thesis, and these were qualitative in nature.[1] In the same way, the evaluation process of the EEM should also emphasise its qualitative nature.

It is appropriate to start this section with a quotation from *"An Open Letter to Cost Model Evaluators"*. Robert Park [Par89] (who is both a developer and evaluator of cost models) argues that an evaluation of a cost model should not be based on the presumption that it is **the models which predict the cost**.

> *The report's focus on model accuracy says, in effect, that you believe that models, not estimators are responsible for estimates! .... Only the most naive estimators would ever turn an estimate over a model. .... The challenge to a model builder is to help the users make correct inputs happen. .... No professional estimator or evaluator should ever encourage anyone to rely on models to make estimates for them. .... The report's occupation with model accuracy seems to be based on an assumption, perhaps made by your sponsors, that an estimating model is some form of precision instrument. An oscilloscope might be a fair analogy. Nothing could be further from truth. The purpose of a model is to help an estimator to perform his tasks, not to do the job for him. It follows, then, that the way to compare models is to evaluate the help they give* [Par89].

Following this line of thought and aiming to get a diverse feedback, the EEM was presented in a number of meetings of Special Interest Groups in Israel, the Netherlands and the UK. These were attended by professional software engineers, IT users, professional managers as well as professional educators in IT and management.

The feedback received was very encouraging. The participants felt that the EEM helps the estimator in understanding what is involved in both the process of software development and the process of estimating the effort required for the development. The way in which the estimator incorporates his decisions into the WBS structure, the ease with which this facility is used when new assumptions or

---

1. See Section 1.10

decisions are made, was of particular interest. Professional educators in IT and management areas expressed their appreciation of the way in which the EEM introduced the decomposition of the SDLC into segments and activities, as well as the way the assumption were recorded and traced. They indicated that the EEM provided a good basis for training managers and IT personnel.[2] Professional software engineers who are associated with the development of integrated tools for software management mentioned that the EEM is applicable for the use in such tools. Furthermore, the decision recording facility should be expanded to additional areas of concern. This facility is applicable only when a bottom-up approach is taken, this is one of the reasons for the EEM being built as bottom-up model.

Additional properties of the EEM were discussed in these discussions, they were mainly associated with the:

- Ability to build interfaces among the EEM and other software management or decision support tools (e.g. PERT).
- Ability to incorporate the EEM into various environments.

These issues were considered when building the EEM, however, it is not possible to provide an indicator of how easy this will be to implement. An interface to PERT tools is of particular interest. The bottom-up approach upon which the EEM is based makes such an interface possible. Interfaces with project management and decision support systems should also be a topic for further research.

The applicability of the EEM to different environments is questionable but not impossible. It is worth indicating that the preliminary findings from the analysis of two projects (a real-time and an embedded environment) suggest that the concept used in the EEM may well be applicable to those environments. The two projects are not included in the sample results below as the projects have not

---

2. Although the WBS is not at all a new concept, managers of software development often start a software planning or estimation session with a blank piece of paper.

yet been completed. The EEM estimates and the project team estimates for the PSD are similar in both cases. The COCOMO Intermediate formulae were used to estimate the effort for one of these projects.

The **second avenue for qualitative evaluation** of the EEM followed the descriptive evaluation of sizing models exerted by the IIT [IIT87a]. Although this evaluation process was directed towards sizing models, most of the compared items are also applicable to effort estimation models. It was felt that such a comparison might add an extra dimension to the evaluation of the EEM. Only features included in the EEM's prototype were considered in this comparison.

The IIT report suggests the following criteria for evaluation: user input, historical data and analysis, methodology, model output and model usability. Each of the criteria will be explained below and then the score earn by EEM is given.[3] The scoring used is 0 - 4; 4 being the best score.

User input

Four issues are dealt within this criteria; user effort while implementing the model, the amount of knowledge and experience required in the application area, training requirements, and the availability of input early in the software life cycle.

- **User effort** is quantified by counting the number of 'input types' to the model and the 'discrete input parameters'. The latter counts the number of values which are required as input for each attribute of 'input type'. The counting is for the following five categories:

  * Qualitative inputs - inputs that are used to characterise the development environment and the complexity of the application.
  * Quantitative inputs - inputs that require determination of information.

---

3. The EEM scoring is the subjective rating of the author of this thesis.

* Identification inputs. These inputs are not used for the estimating process.
* Modular and functional input - inputs that require knowledge of the modularity and functionality of the software system.
* Calibration factors - correction factors that reflect a development environment which could cause deviation from an estimate for a 'typical' development environment.

The EEM would have a low score using the IIT criteria as it requires substantial of input. However, it should be noted that each of the models evaluated by the IIT asks for detailed information, but groups the individual attributes into categories and each category is counted as an 'input type'. For example, the counting rules for the 'input types', called 'external inputs', are defined to include the following input types: inputs files; tables; input forms; input screens and input transactions, all of which cross the external boundary of the system and causes processing to happen. In the EEM an equivalent to this categorisation could be grouping of the size attributes into categories of the cost drivers. This procedure would have greatly reduced the number of input types used in the EEM.

- **Amount of knowledge required, training requirements and availability of input early in the software life cycle.** The basic knowledge, that is required to apply the EEM early in the life cycle, is available in the organisation. It is part of the deliverables of the 'preliminary survey'. The EEM tries to be a self explanatory, the input is generally easily understood and therefore it can be used by inexperienced estimators. There are few training requirements. Therefore, if the EEM were to be included in the IIT evaluation the following ratings would have been scored:

* Can the model be applied without knowledge in the application area? Rating 3.
* Are the inputs easily understood? Rating 4.
* Is the input available early in the SDLC? Rating 4.

Underlying methodology

The evaluation questions for this criteria are:

- **Is the model a 'white box'?** Is the underlying theory in the public domain? The EEM is based on the stages of the software development process in association with the standard of effort. It is a open framework. It would have scored 4, for this criteria.
- **Is the model applicable to different user environments?** The particular issue addressed here is the applicability of the sizing model to scientific and real-time systems. The applicability is judged by the size results yielded by a particular model. To date the EEM has not concentrated on this area. Function Point (FP) based models, for which the IIT survey has not been able to attain access to published studies that address this finding, were rated as 2. It is assumed that the EEM falls in the same category and would have a similar rating.
- **Are the equations parametric-based?** This criterion is concerned with the procedure in which the model's parameters were obtained. Only two options are considered: models which were developed and based on project data are scored as 4, and those which are derived purely statistically are assigned a rating of 0. The EEM is based on the history of project data, but heuristic procedures are also involved in the process of building it. Therefore it is likely to have a high score (3).
- **Is the model's result accurate?** This issue is addressed in the quantitative analysis of the sample data (see Section 8.3 and Paragraph 8.3.1)

Model output

The third criterion concerns the model output. The questions in this section are:

- **Is there a probability associated with the estimates?** The EEM does not

forecast probability ranges. Nevertheless, the EEM collects the minimum, maximum and most likely values for the appropriate quantitative inputs, from which a range of estimates incorporating probability can be calculated. This approach substitutes the preliminary approach taken in the design of the EEM, as expressed in the early versions of the questionnaire. The estimator was asked to associate one of three degrees of uncertainty (high, medium or low) with the input metrics. The idea was to associate with each uncertainty level a range of probabilities.

- **Does the model estimate Function Points?** All the FP based models are assigned a high rating of 4. For the purpose of this evaluation it can be assumed that the EEM would have scored 4.[4]
- **Are inputs summarised on the output report?** The concern here is whether all the input data is provided in the output summary reports. The EEM provides a full report of the size and the complexity assumptions. It would have scored 4.
- **Does the model provide graphic capabilities?** The EEM provides only limited graphic capabilities in its prototype version. It would have been therefore scored 1 or 2.

Model usability

Model usability is the final category which is assessed in this survey. Only one of the two questions included in this category is relevant to the evaluation of the EEM.[5] It is:

- **Does the model have a user-friendly interface?** The individuals who were exposed to the EEM rated its interface highly, although the EEM is

---

4. It should be mentioned that the sizing models generally incorporate the use of a conversion table from FP to LOC. As the EEM is not a sizing model this is not relevant to the evaluation, therefore the score 4 was assigned.

5. The second question is related to the availability of user support

demonstrated as prototype which was built for research purposes only and as such does not include all the help and explanations which would be required. It is assumed that the IIT would have scored the EEM for user friendliness as 4.

8.2.2 Evaluation summary

This comparison reinforces the quality of the EEM. It performs at least as well as the established models included in the IIT sample. Further analysis shows that:

* In six of the twelve assessment criteria none of the models scored better.
* In one of the criteria only 9% of the models presented a better score than the EEM.
* In the five remaining assessment criteria around half of models include in the IIT sample scored better then the EEM.

The EEM shows strengths in each area included in the evaluation survey. The comparison of the EEM with the sizing models included in the IIT survey is presented in Table 8.1

ASSESSMENT CRITERIA	NUMBER OF MODELS AS SCORED IN THE IIT SURVEY RATING					EEM RATING	THE EEM ACHIEVED A BETTER SCORE THAN THE IIT MODELS (%)
	0	1	2	3	4		
USER INPUT (the sample includes 11 models)							
Are there few inputs to derive?	6	0	1	1	3	0	54
Can the model be applied effectively without knowledge experience in the application area?	1	3	1	5	1	3	9
Are input easily understood?	0	1	4	1	5	4	0
Is input data available early in life cycle?	0	0	2	4	5	4	0
HISTORICAL DATA AND ANALYSIS	**Not Applicable**						
UNDERLYING METHODOLOGY (the sample includes 11 models)							
Is the model a white box?	6	0	2	0	3	4	0
Is the model applicable to different users environment?	0	1	4	0	6	2	54
Are the equations parametric-based?	4	0	0	0	7	3	64
Did the model provide accurate results in a case study?	**This issue is addressed separately.**						
MODEL OUTPUT (the sample includes 9 models)							
Is there a probability associated with the estimates?	3	0	0	0	6	0	67
Does the model estimate Function Points?	5	0	0	0	4	4	0
Are inputs summarised on the output report?	2	0	0	0	7	4	0
Does the model provide graphic capabilities?	5	0	1	0	3	1	44
MODEL USABILITY (the sample includes 9 models)							
Does the model have a user-friendly interface?	0	0	3	4	2	4	0
Is user support available?	**Not Applicable**						

Table 8.1 A comparison of the EEM with the sizing models included in the IIT survey.

8.3 EVALUATING THE EEM - QUANTITATIVE ANALYSIS

The evaluation sample included 18 projects from heterogeneous environments. The projects differed in size, programming languages and technology used to develop them, the type of organisations they served and the contractors' organisations which built them. It should be noted that a few of the projects originated in software houses where the actual effort was limited, by definition, to the 'estimated effort'. Under these conditions an average comparison is not the best test. However, it should give us some indication of the accuracy of the EEM prediction. Two questions should be asked:

* Does the EEM yield satisfactory planning approximations?
* Can the model improve the teams' original estimates?

Conte et al [Con86] suggest the following measures should be used to evaluate effort models for accuracy: the *MRE, MMRE* and the *PRED(level)*.[6] They will be analysed below. The sample data produces the results as shown in Table 8.2.

The EEM results under the Mean Magnitude of Relative Error *(MMRE)* criteria is 36.1. This is considered to be a satisfactory result for a sample built as heterogeneously as the EEM, although Conte considers only $MMRE <= 25$ as acceptable for effort prediction.

However, even when the *MMRE* is small, there may be one or more predictions that are very poor. Therefore, Conte et al suggest an additional criteria for the goodness of the prediction. This is the *PRED* at a specified level, which indicates the percentage of predictions that are equal to or less than the *PRED* specified level. This means that only the *MRE* values which are equal to or less than the acceptable level are considered in this calculation.

---

6. See discussion in Paragraph 5.2.4

ID	PROJECT DATA Actual* total effort	Estimated total effort	EEM coarse estimates effort	PSD estimates effort	MRE project teams	EEM	KLOC
r	350000	32518	225591	54142	90.7	35.5	2000
w	16800	12600	32333	3880	25.0	92.4	1000
n	25000	25000	38880	12506	0	55.5	750
x	2600	2000	8383	1509	23.1	222.4	175
o	40572	19759	20285	1420	51.3	50.0	162
p	40577	35525	43853	11402	12.4	8.1	156
h	4590	2750	3800	456	40.0	17.2	200
a	2930	3150	3118	530	7.5	6.4	80
c	7500	**	6443	451	**	14.1	80
t	25000	23000	29840	5968	8.0	19.3	80
g	46368	24192	49040	12260	47.8	5.8	70
q	8600	5000	9720	972	41.8	13.0	65
l	7900	3000	12387	4955	62.0	56.8	60
j	16059	14895	14520	4356	7.2	9.6	46.6
d	8547	6527	9033	2439	23.6	5.7	36.5
f	3213	3066	2652	345	4.6	17.4	30
i	2200	1800	2590	518	18.2	17.1	25
b	3108	2020	3260	652	35.0	4.9	19
PRED(0.25) =		55.5		66.7 ***			
PRED(0.20) =		38.9		66.7 ***			
PRED(0.15) =		33.3		44.4 ***			
MMRE		29.5 ***		36.1			

* Effort is given in man weeks
** Not estimated. The *MMRE* was considered for the *PRED* calculation.
*** Best under this criteria.

Table 8.2 Comparison of the EEM coarse estimates and estimates with the actual and estimated effort as provided by project teams.

Examining the results for the *PRED(level)* shows that the EEM predicts the same for *PRED(0.25)* and *PRED(0.20)*. These results indicate that 66.7% of the coarse estimates in the sample are within 0.20 of the actual effort. Although the

acceptable value suggested by Conte is *PRED(0.25)* = >75%, the EEM results could considered satisfactory results under the sample restrictions. Putting these results in perspective the reader is referred to the previous discussion on the topic of 'diseconomies of scale' and the impact of inaccuracy in size estimates on the estimates of effort.[7]

8.3.1 Comparison: the EEM planning approximates with the projects teams' estimates

The estimates of the project teams and the coarse estimating resulting from the EEM are compared for accuracy. The criteria used for this comparison are the *MMRE* and the *PRED*. The *PRED* is tested for three specified levels 0.25, 0.20 and 0.15.

A comparison of the EEM results with the estimates of the project teams shows that their estimates are better than the EEM under the Mean Magnitude of Relative Error *(MMRE)* measure, as the smaller the value of the *MMRE*, the better the prediction. Examining the results for this measure shows that the EEM predicts better than the project team at a prediction level of 0.25, though this is not statistically significant. When the *PRED* level is reduced to 0.20, the EEM performs markedly better than the project teams ('as a collective') in estimating the total effort. In this case only 38.9% of the estimated values fall within 0.20 of their actual effort, whilst with the EEM 66.7% did so.

Despite the restrictions indicated above, the conclusions that can be drawn from this sample are that:

- The EEM coarse estimates are as good as the average estimates produced by the 'collective'.
- The EEM performs considerably better than the 'collective' under *PRED* when relatively low values are specified for it.

---

7. See Paragraphs 3.6.1 and 4.1.1.

8.3.2 Regression Analysis

The second qualitative avenue taken to evaluate the EEM is linear regression analysis. This test is used to indicate the extent to which the various sets of estimates and the actual effort are linearly related. A high value of R^2 suggested either that a large percentage of variance is accounted for, or that the inclusion of additional independent variables in the model is not likely to improve the model dramatically.

Two sets of linear regression were performed by using the actual effort (measured in PH) as the dependent variable and the estimated effort (PH), given by the EEM and by the projects teams, as the independent variable. The first regression set aims to compare the coarse estimates yielded by the EEM and those produced by the projects teams. The second set aims to measure the correlation between the EEM estimates for the effort required for the PSD with the actual effort. The results of the regression analysis are given in Table 8.3.

	Teams estimates	EEM coarse estimates	EEM PSD estimates
Constant	9184.042	6730.816	1410.738
Std. Err. of Y Estimates	9869.875	9053.914	3002.500
R Squared	**0.331994**	**0.971071**	**0.946406**
X Coefficients	0.084192	0.634651	0.153670
Std. Err. of Coefficient	0.029856	0.027385	0.009083
Excluding observation No. 1			
Constant	612.3531	2163.707	
Std. Err. of Y Estimates	4671.161	8007.16	
R Squared	**0.826776**	**0.766112**	
X Coefficients	0.664747	0.943979	
Std. Err. of Coefficient	0.078563	0.134671	
No. of observation	18	18	18
Degrees of freedom	16	16	16

Table 8.3 Regression analysis

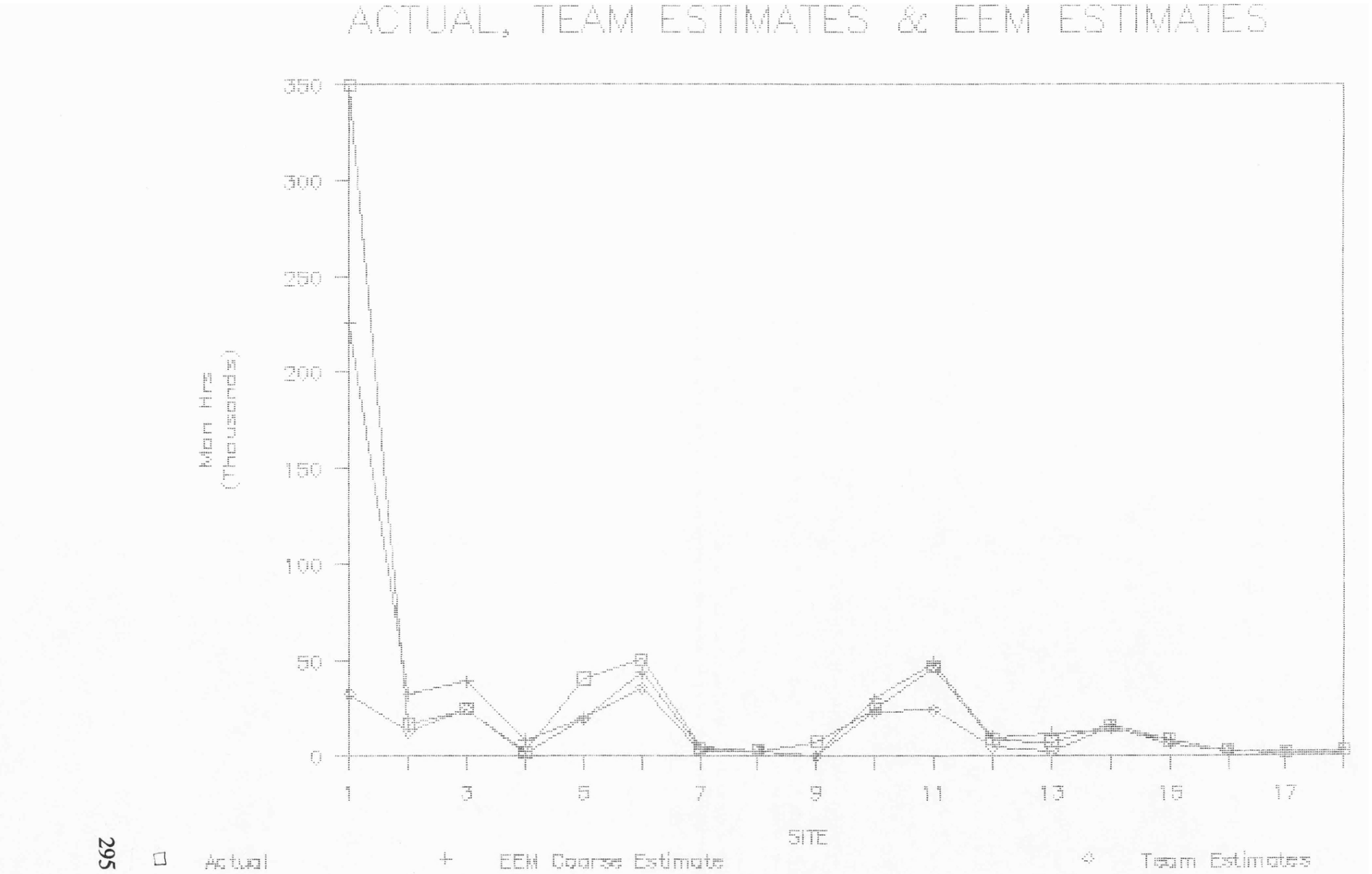
The results of the regression analysis show the EEM estimates are highly correlated with the experience. The EEM coarse estimates resulted either an R^2 of 0.97 and with a 'standard error of estimates' of 9053.9, while the projects teams 'as a collective' estimates resulted with an R^2 of only 0.33 with a 'standard error of estimates' of 9869.8. This test suggests that the EEM estimates, for the sample data, are about **three times better** than the projects team's estimates. However, since the data set includes one extreme observation, an additional regression analysis was performed excluding this observation. This resulted with an R^2 of 0.76 and with a 'standard error of estimates' of 8007.16 for the EEM coarse estimates, while the projects teams prediction was correlated with the actual experience with an R^2 of 0.82 with a 'standard error of estimates' of 4671.16. The teams present a slightly better correlation in this case.

The second regression was implemented by using the EEM estimates for the PSD as the independent variable. This resulted with an R^2 of **0.94** with a very low 'standard error of estimates', namely **3002.5.** Once again suggesting the EEM estimates are approximately **three times better** than those of the project teams.

A plot of the regression results is shown in Figure 8.1

8.4 CONCLUSIONS

The EEM presents strength in both the quantitative and the qualitative aspects of effort estimation. In half of the qualitative criteria assessed by the IIT survey **none** of the models scored better. The areas in which the EEM scored low are those which were not part of the research or were not implemented in the prototype and thus could not be evaluated. e.g. incorporating probability into the EEM or the applicability of the model to different users' environments. The quantitative results of the EEM are also encouraging. This is despite the heterogeneous nature of the sample data evaluated by the EEM.


ACTUAL, TEAM ESTIMATES & EEM ESTIMATES
Man Hrs (thousands)
350
300
250
200
150
100
50
0
1
3
5
7
9
11
13
15
17
SITE
Actual
EEN Coarse Estimate
Team Estimates

Chapter 9
CONCLUSIONS

9.1 INTRODUCTION

This chapter summarises the fundamentals of the EEM and its unique features. The conclusions reached from the analysis of the data collected, the use of the prototype and from the demonstration sessions held with various professional groups will be shown. The chapter finishes with an agenda for further research.

9.2 EFFORT ESTIMATION MODELS FOR SOFTWARE DEVELOPMENT

Most of the parametric models offer estimates for effort and duration, and some deal with risk assessment. The output style and content vary between the tools. Some of the tools appeal to IS/DP personnel while others are preferred by those oriented towards engineering. Human interaction and judgement are essential throughout. The level of confidence in the estimates differs according to the stage in life cycle at which the estimates are made, and the amount of solid information known at that time.

The models offer either a basic formula for the nominal effort in PM based on productivity to which a correction multiplier is applied to account for

complexity factors believed to impact on the productivity of software development. Alternatively a formula which describes the observed behaviour of effort accumulation throughout the life cycle of the software development is presented.

The first type of model is derived empirically and the second group is analytically based. Many of the estimation models are composites of analytical, empirical and heuristic approaches.

Productivity is measured in Lines of Code or Function Points. The LOC measure assesses the size of the completed product while FP aims to measure the amount of functionality incorporated into the product. The units of measure used here are the number of files, inputs, outputs, reports and screens in the target system. The correction factors affect one or all phases of development.

There is no agreed standard set of rules for counting the LOC or the FPs. This also applies to the assessment of the correction factors.

9.3 WHAT IS UNIQUE ABOUT THE EEM IN RESPECT TO CURRENT MODELS AND TOOLS?

The EEM approach adopted in this thesis and the unique features incorporated in the EEM will be summarised below.

This thesis has argued that, at the outset of the project life cycle, estimates can be developed with a high degree of accuracy only for the PSD phase. Therefore, the **EEM estimates the effort required for the Preliminary System Design** and from that estimate, extrapolates coarse estimates for the total project development. The coarse estimates are extrapolated according to resource allocation among phases of development, as known statistically and by the estimator judgement about the foreseen distribution of a particular project. The coarse estimates are updated when uncertainty is reduced.

The EEM is a complementary approach to the current models for estimating the effort required for software development. The EEM emphasises the estimating process early in the software life cycle. The estimates are developed for a single phase. The model should be implemented in tandem with

other models which are built to estimate the effort when the requirements are completed and the software product is specified. None of the current models or tools take this approach. Nor do most of the models estimate the effort required for the feasibility and the requirements, they are only used after the design of software product is completed.

The information available to support the estimation process early in the life cycle is limited. The uncertainty associated with this information in the product and development process is high. **But estimates are needed.** They are fundamental for evaluating and choosing among alternatives, for planning and budgeting the development of a software product. This is the main source of the estimator's dilemma, to which the EEM provides an answer. However, only a certain level of accuracy and precision is possible at the early phases of the life cycle, when there is a minimal level for knowledge of what the software is to do.

Thus, the approach taken in the EEM emphasises the **method** of achieving the estimates, the **process** in which the estimator is led through the estimation session, and his **ability to influence** the results, not only by estimating the size components and assessing the complexity level of the development process, but also by changing and adding basic components, if required. It is not the model which produces estimates, it is the estimator. The EEM is an **open framework** which offers a set of standards, advice when needed and an estimation process.

Estimating the effort for software development requires **quantitative** and **qualitative** information about the product to be developed and the process in which it is being developed.

In **quantitative product** information, items like the number of reports, screens, outputs and inputs in the target system are considered. In the **process information,** quantities of items which consume resources, but which do not affect the product size are considered. These are items associated with the strategy chosen for the development process, e.g tenders that have to be evaluated and contracts negotiated and signed; size attributes which are associated with the replaced system (either manual or automated system), or size items that affect, for example, the capacity planning and the project organising activities. Although the items which are considered here as quantitative information are unique to the

process of the development of the PSD, they will impact upon the resources consumed further in the development. Using these units of measure for the project size is unique to the EEM.

Qualitative information is required for both the product and the process. The product structure, the logical and structural complexities, the inter-connections among processes and/or functions are considered as **product qualitative** information. Although deterministic metrics are offered for these attributes, at the outset of the development there is not enough knowledge to allow the quantitative assessment of these of complexity attributes. The only information available on this matter is the estimator's judgement about the number of complex functions from the user's viewpoint and the anticipated number of corresponding complex processes from the technical point of view. The **process information** is associated with the dynamics of the environments in which the development takes place.

The **complexity and risk indicators** are assessed according to these sources and types of information. The approach taken in the EEM is that the complexity and risk cannot be measured in a rigorous way early in the project development. The estimator can, however, use his knowledge, experience and judgement to score each complexity and risk determinant. The **accumulation** of these scores indicates the level of complexity and the risk anticipated in the development of the project in an ordinal measure.

The **bottom-up approach** chosen for modelling the process of estimating the effort required for software development links both the process of building a software product and that of estimating the effort required for it. These links afford the foundation upon which the EEM is established and from which the infrastructure for a process model for effort estimation can be built. Furthermore, this link provides an insight into the process which is being modelled. It is assumed that a better **understanding** of both the software development process and the effort estimation process will help in producing better estimates, since the major problem is underestimating the effort, caused by not knowing what is involved in a specific solution.

The EEM links these two processes through the **activities** which encompass

the PSD development process. The connection is implemented in two different ways:

- Each of the activities is associated with one or more **cost drivers** to which a **'standard of effort'** is associated, based on the complexity assessment of the system. In other words the **productivity** is measured for each of the cost drivers affecting an activity, at three various levels of complexity.
- Assumptions (e.g. quantities associated with the various cost drivers) and decisions taken throughout the estimation processes are recorded and linked to the relevant activity or segment. This allows the incorporation of a feedback mechanism into the EEM

9.4 THE BENEFITS OF THE APPROACH TAKEN IN THE EEM

A number of benefits emerge from these features and they are summarised below.

The EEM emphasises the **process of estimating** the effort for software development. The model directs the estimator through the estimation session; the process is understandable and therefore easy to change and to adapt to an individual organisation. This was clearly demonstrated using the prototype tool developed. The tool was well received in each of the demonstrations held.

The EEM uses information which usually exists in the organisation at early stages of software development as a result of the 'preliminary survey' and information residing in the EEM knowledge-base. However, this information is not always sufficient to provide estimates without consulting the estimator and requesting him to estimate the effort required for particular functions, which vary widely among projects. The EEM supports this consultation session by providing information on past projects.

The use of the bottom-up approach allows the building of an integrated tool for software management which incorporates both the software development and effort estimation processes. For example, by providing effort estimates for each activity, and using the base model, particularly the data suggested to be

associated with each activity, it is possible to interface the EEM with a PERT or other project management tool. In addition, based on the estimates and the knowledge-base, the model can provide the project manager with advice on topics such as alternate schedules, milestones and required products at the end of each milestone (end of phase and/or segment and/or activity). At a later stage, it can provide guidance for choosing the appropriate model for effort estimation based on the estimation objectives, type of system, etc.

Basing the estimates on activities and cost drivers is more understandable and approachable for project managers and analysts, than a formula which is based on LOC or FPs. They can easily understand why the estimates resulted. If the managers do not agree with the results and/or the components, they are allowed to change the components associated with the process as an ad-hoc change. The EEM is an **open framework** understandable to all parties involved in the process of estimating the effort, the users, managers and IT personnel, in which measurement and judgement are incorporated.

Recording the assumptions (e.g. quantities associated with the various cost drivers) and decisions taken throughout the estimation processes and linking them to the relevant activity or segment, enables us to incorporate a feedback mechanism into the process of software development. Since the activity is the basic unit for many of the processes associated with the management of software development, this foundation can be expanded into a process model. Consider an example: a message indicating the completion of an activity is sent and incorporated into the process model. This message initiates an automated measuring activity of the quantifiable cost drivers associated with the completed activity. Since a cost driver is associated with more than one activity, this will point out a specified deviation from original estimates and will suggest corrective measures for the estimated effort (or a design-to-cost).

The EEM inputs provide information which allows adjustments to the estimates made. For example: differentiating between novel and familiar application areas. These inputs affect the complexity assessment of the project and thus the 'standard of effort' used in calculating the estimated effort, which is the prime factor affecting the project schedule. However, it may be useful to

provide the estimator with a set of rules which are of particular interest to the project for his additional consideration of schedule and/or effort adjustments. When analysing the data received through the questionnaires, such rules were applied. This could be easily added into the knowledge-base, automated and offered to the estimator when appropriate.

The EEM can also be used as a training tool for effort estimation. It will help managers and analysts understand factors that are likely to influence the effort estimation process.

The quantitative and qualitative evaluation of the EEM (see Sections 8.2 and 8.3) indicates that the EEM produced as good estimates as the sample data and shows strength in the majority of the qualitative criteria.

9.5 AGENDA FOR FURTHER RESEARCH

The contribution of this thesis to further research is based on:

- The infrastructure presented and demonstrated in the EEM. The concept developed, which was partially implemented in the prototype, can be further developed to become a complete process model for managing the development of software. Incorporating the EEM into a process model is feasible, however, further research is needed.[1]
- More automation. An example could be the adjustment of the effort associated with a segment as result of the assessment of the complexity rules. An additional example is the incorporation of a feedback mechanism into the EEM
- Developing a descriptive language to be used for comparison of project attributes.

---

1. The term: 'process model' is still not well defined in the literature, yet it is commonly used to describe a model for the whole process of software development.

The advantages highlighted earlier can be developed further. For example, **recording the decisions** provides the foundation for **learning capability.** This research suggests a way to incorporate assumptions and decisions into the process of estimating and linking them into the life cycle. This facility can be expanded to account for decisions taken in the design process.

Estimates are based on comparisons. Differences and similarities to past experiences are the basis for expert judgement and analogy. Therefore, there is a need to acquire quantitative information from projects developed in the past to put the comparative evaluation on a demonstrably sound footing. **Incorporating recorded assumptions** taken in addition to the quantitative information helps in this purpose and should be considered in the design of historical data-bases. There is a need to assess further the value of recording the additional assumptions suggested.

Furthermore, by taking advantage of AI, the EEM could 'learn' from the data generated by active projects and incorporate that experience into the knowledge-base.

The EEM records the description of the activities and the forecast effort suggested by the estimator. An effort should be made towards providing a descriptive language for comparing or contrasting the content (description) of these activities. The same is applicable for the recorded assumptions and decisions. This will help in building traceable trails for reasoning and explaining the estimated results, and thus communicating the credible estimates to management.

The ability to incorporate the EEM into various environments should also be a topic for additional research. Although this thesis focused on estimating the PSD for business software applications, the effort could be expanded to embedded, real-time systems and basic software. It might be the case that additional cost drivers will be required and others should be eliminated. There is a need for further research in the area of resource allocation.

A continuous effort towards the incorporation of a wider range of development strategies and additional types of software projects in the EEM, would be of real value to the industry.

9.6 CONCLUSIONS

The EEM aims to support the process of the economic evaluation of alternatives for software development early in the life cycle. It aims to serve all the parties involved in the process of managing the software development and hence are in a need for coarse estimates of the effort required for the development. The motivation in developing the EEM is the need to establish an 'open framework' for the estimation process. The term 'open framework' indicates a process which is understandable to all parties involved in building and managing software development and, thus, in estimating its required resources. Each of the individuals involved in the process can use and is authorised to use the model. The user can input his view of the application domain and of the surrounding environments, he can analyse the estimates resulting from his model of the application and environment domains. When the basic assumptions do not fit his own judgement, they can be changed or amended. However, such a change is considered by the EEM as an ad hoc change only, it is not incorporated in the EEM's knowledge-base. In cases where additional research is required before estimates can be suggested, the user can consult the EEM for analogy to previous projects. The underlying assumption is that the practice of estimating can be improved - learned, by the individual who is using it. However, understanding can be gained only if the process of estimation is tied to the process of software development. The conceptual design of the EEM provides such a link.

The results are that the EEM showed strength in both the qualitative and the quantitative aspects as discussed previously.

APPENDICES

Appendix 4A Definition of the Information Domain - Function Point Analysis.

Number of user inputs.
Each user input that provides distinct application oriented data to the software is counted. Input should be distinguished from inquiries, which are counted separately.

Number of user outputs.
Each user output that provides distinct application oriented information to the user is counted. In this context output refers to reports, screens, error massages etc. Individual data items within a report are not counted separately.

Number of user inquiries.
An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files
Each logical master file (a logical grouping of data that may be part of a large data-base or a separate file) is counted.

Number of of external interfaces.
All machine readable interfaces (data files on tape or disk) that are used to transmit information to another system are counted.

Appendix 5A. A Comparative table - Factors affecting productivity of software development and complexity determinants.

N-SDC;
A-ARON;
D-Doty;
AL-Albrect;
S-SLIM;
J-Jensen;
E-ESTIMACS

FZ-FARR AND ZAGORSKY;
T-TRW;
W-Walston and Felix;
DM-DeMarco;
C-COCOMO;
R-RCA-PRICE-S;

SIZE ATTRIBUTES

	N	FZ	A	T	D	W	AL	DM	S	C	J	R	E
-No.of instructions	-	FZ	A	T	D	W	AL	-	S	C	J	R	-
- '%' of new instruction	-	FZ	-	T	-	-	-	-	S	C	J	R	E
-No. of documentation	-	FZ	-	-	D	W	-	-	S	-	J	-	E
-No. of data elements	-	FZ	-	-	-	W	AL	DM	-	-	C	-	-	-
-No. of user inputs	-	-	-	-	-	-	AL	-	-	-	-	-	E
-No. of user outputs	-	-	-	-	-	-	AL	-	-	-	-	-	E
-No. of user/on-line inquir.	-	-	-	-	-	-	AL	-	-	-	-	-	E
-No. of internal Log.files	-	-	-	-	-	-	AL	-	-	-	-	-	-
-No. of master files.	-	-	-	-	-	-	-	-	-	-	-	-	E
-No. of external interfaces	-	-	-	-	-	-	AL	-	-	-	-	-	E
-No. of Function Primitives	-	-	-	-	-	-	-	DM	-	-	-	-	-
-No. of personnel	-	-	A	-	-	-	-	-	S	-	J	-	E

PROJECT ORIENTED COMPLEXITY

-Type of application	N	-	A	T	D	W	AL	DM	S	C	J	R	E

at system - level

e.g.:Business/non-business
Real-time/non real-time

-Duration	-	-	A	-	-	W	-	-	S	C	J	R	
-Type of program	N	-	A	T	D	W	AL	-	S	C	-	R	-

sub-system / module level

Program categories
(e.g: TRW and or Aron's
categorisation
% mathematics instruction
No.of sub-programs
Stand alone program)

-Language	N	FZ	A	-	D	W	-	DM	S	C	J	R	E
-Reuse	-	FZ	-	T	-	-	-	DM	S	C	J	R	E
-Required reliability	-	FZ	-	-	D	-	-	DM	S	C	J	R	E
No. of major subsystems	-	-	-	-	-	-	-	-	-	-	-	-	E
Novelty of business function	-	-	-	-	-	-	-	-	-	-	-	R	E
Novelty of system	-	-	-	-	-	-	-	-	-	-	-	R	E

ORGANIZATIONAL ENVIRONMENT

-Customer interface complexity	N	-	A	-	D	W	AL	-	-	-	-	-	-	
-Customer originated changes Lack of requirements Requirements volatility Stability of design Availability of project objectives analysis	N	-	-	-	D	W	AL	-	-	C	J	R	E	
-Multy-site	N	FZ	-	-	D	-	AL	-	-	-	J	R	E	
-No. of company functional organisation.	-	-	-	-	-	-	-	-	-	-	-	-	E	
-No. of people in organisation involved	-	-	-	-	-	-	-	-	-	-	-	-	E	
-Different host target H W	N	-	-	-	D	-	AL	-	-	-	J	R	-	

TECHNICAL ENVIRONMENT

-Modern programming practices Use of software tools and techniques	-	-	-	-	D	W	AL	-	S	C	J	R	-	
-Communcication and disistributed systems	-	-	-	-	-	-	AL	-	-	-	-	-	E	
-Logical complexity	-	-	-	-	-	-	-	-	-	-	-	-	E	

-Computer attributes

-Hardware configuration concurrent hardware development	N	-	-	-	-	-	-	-	-	-	-	R	-	

-Computer access	-	-	-	-	D	-	W	-	-	S	C	J	R	-
-Time constraints	-	-	-	T	D	W	AL	-	S	C	J	R	-	
-Space constraints	N	-	-	-	D	W	AL	-	S	C	J	R	-	
-Hardware devices Random access device used computer resources	N	-	-	-	D	W	-	-	-	C	J	R	-	
-Schedule Constraints	-	-	-	-	-	-	-	-	S	C	J	R	-	
-Criticality of data trafic load and performance	-	-	-	-	-	-	-	-	-	-	-	-	E	

PROJECT TEAM COMPOSITION

-Application experience	-	-	-	T		-	W	AL	-	S	C	J	R	E
-Continuiity	-	-	-	-		-	W	-	-	-	-	-	-	E
-Lnaguage experience	-	FA	-	-		D	W	-	-	S	C	J	R	-
-Capability	-	-	-	-	-		W	-	-	S	C	J	R	-
-Hardware experience	N	-	-	-		D	W	-	-	S	C	J	R	-

Appendix 6A. The EEM Questionnaire

EFFORT ESTIMATION

MODEL

ZEEVA LEVY

INFORMATION SYSTEMS SUB-DEPARTMENT

The London School of Economics and Political Science
Houghton Street, London WC2A 2AE
Telephone: 01-405 7686 / ext. 2958
Telex: 24655 BLPES G
Facsimile line (01) 242 0392

September 15, 1989

Dear Project Manager,

I am currently undertaking research into Effort Estimation for Software Project Development (EEM) at the London School of Economics (the Information Systems Sub-department) and at Imperial College (the Management School). I am asking experienced project managers such as yourself, to assist me by answering a questionnaire. All information provided will be treated as confidential.

The EEM for Software Project Development I have been developing, aims to estimate, at the outset of a software development project, the person months of effort required for the Preliminary System Design (requirement specifications and product design) of software development project. The incentive for my attempt to develop this model stems from my twenty five years of accumulated experience in the process of software development. Through these years I have been exposed to the need for estimating, evaluating and justifying the development of software projects. I had the opportunity to observe closely the ways in which development projects are being customarily evaluated and justified. While working in the U.S.A for several years , I learned about relevant approaches which I qualified to my personal usage and improved to suit my own environment. The EEM proposed here, is based on the accumulated experiences in this area and it is a direct product of it, which I would like to share with you while working for my Ph.d.

I would appreciate it if you would be kind enough to take the time to read the two parts of the attached document. Secondly, could you please think about a project you were recently involved in and for which you could assemble, the information relating to the project profile and to the complexity of the project environment. The information I ask for is in Part 2 of the attached document. Thirdly, could you please answer the questionnaire.

Your answers will be then processed by the EEM. The estimates produced by the model will be analyzed and compared with your data relating to the actual effort needed for the development of your project and the original estimates. This process would be of help in both analyzing the importance of each of the questions to the estimation process and in the evaluation of the model and its' fine tuning. It might also enable me to establish a historical database for estimating software project development.

I am available for further information, explanation of for facilitating the process of answering this questionnaire. If you require any help or need any further information, please do not hesitate to contact me at London School of Economics, telephone number:
(01) 405-7686 / ext. 2958 or, at home: (01) 4856277.

Finally, I would like you to mail your response by the mid of October, 1989 to:
Mrs. Zeeva Levy ,
Information systems sub-department,
London School of Economics.
Houghton Street,
London, WC2A 2AE. Telex: 24655 BLPES G Facsimile line (01) 242 0392

Thank you very much for your co-operation. When the project is completed, I will be pleased to send you the results and would be interested in your observation.

Sincerely,

Zeeva Levy.

EFFORT ESTIMATION FOR SOFTWARE PROJECT DEVELOPMENT

TABLE OF CONTENTS.

PART 1

Introduction.

There exists the need to estimate the effort (person days/months) required for software project development towards and during the development process. Unfortunately, it is difficult to estimate this effort. Much of the difficulty stems from the degree of uncertainty associated with both the project oriented problems and the inherent complexity of managing the software development process. Whilst uncertainty exists in each of the stages where the estimates are needed, it is much higher in the first stages of the software project development process.

The objectives of the *Effort Estimation Model (EEM).*

The *Effort Estimation Model for Software Development Project* is a support system for the estimation process, which takes place when a project is about to start, and uses techniques more consistent with the level of current knowledge generally available in the organization at that time. More precise estimates for the effort required at the later stages of the software development life cycle, are done in a more detailed form, by using different techniques, towards the end of the requirement analysis and the preliminary design stages. Different estimating approaches may be used at different points in the system development process.

The *Effort Estimation Model (EEM)* emphasizes the estimation process at the outset of the project development process. It is done as a part of the economic evaluation which at that point of time is done only at a low level of precision, that enables the management to determine the project feasibility.

The *EEM* aims to:-

- Serve the project team members, users and management. It does so by providing its users with a guide for action by illuminating both the process associated with the software project development and that associated with the production of estimated effort.

- Assist experienced project managers and all other data processing professionals by suggesting an interactive and structured estimation process which facilitates thinking about both their work and decision making.

- Serve as a training tool for the inexperienced project manager and user, by proposing a standard procedure for software project development and for the estimation process. It does this by:-

  -- Providing the project leader with a choice of development strategies, each of which is decomposed into relevant, manageable and functional units and activities.

  -- Presenting the potential decisions in each of the relevant stages or the actual decisions taken during the estimation process.

- Provide a **basis for:**-

-- Assessing project risk.

-- Comparing and evaluating the various development alternatives.

-- Developing a working plan for the project's next stage.

These qualities allow informal interaction among all parties in the development process: the user, the project manager, the project team and the organization's management, and hence improves reliability, produces more precise estimates and decreases unplanned overruns.

Who is the *Estimator?*

The estimation process can be implemented by an individual or a group of estimators. The group should be composed of the project team members or of various data processing personnel, experts from the user's department or external consultants, who use their best judgement in order to determine the different aspects of the project profile. A group can work in one of the following ways:-

1. Each group member operates the model separately and eventually the model proposes estimates based on the data provided by all members. In this way the members act individually and the model computes an average of the group estimations based on the weighted answers of the team members.

2. The group members arrive at an estimate to a given question and only one agreed answer updates the model. Here the group can be assisted by the Delphi decision process in order to achieve consensus among the members, or by a group decision process.

TERMS AND CONCEPTS.

While operating the *EEM,* the estimator may come across some terms and concepts which will be clarified here:-

1. The Software Development Life cycle.

The *Effort Estimation Model* is based on the assumption that the project development process is supported by a standard software development process, which serves as a tool for directing the creation of new information systems. It is customary to consider the software development process as having a life cycle, characterized by a top-down approach of breaking the process into manageable, logical and functional units. The development process is decomposed into phases, each having defined starting and ending points. Each of the phases is further decomposed into individual work segments, each of which produces pre-defined end products and aims to achieve a specific target. Each of the segments contains a group of standard controllable activities.

The *Effort Estimation Model* uses a consistent set of terms to describe this conceptual top-down approach:-

- A *phase* is a major self-contained component. Three **phases** encompass the development process:-

-- Project Planning.

-- Preliminary System Design.

-- Detailed Design and Implementation.

- A *segment* is a logical part in accomplishing the objectives of the particular phase.

The Preliminary System Design phase, includes the following **segments**:

-- Organization.

-- User requirements.

-- Technical design.

-- Technical support.

-- Implementation schedule.

-- Cost / benefit analysis.

-- Management review and approval.

-- Hardware and software direction.

-- Application software evaluation and design.

-- Hardware and software selection.

- An *activity*. Each segment contains a group of standard *activities*, which provides the project team with guidelines to accomplish the segment end result.

For example, the Project Definition segment contains **activities** such as:-

-- Project initiation.

-- Review present status.

-- Identify business objectives and information strategies.

-- Survey information need.

-- Identify hardware and software environment.

-- Develop conceptual design.

-- Investigate application software alternatives.

-- Evaluate development alternatives.

-- Prepare project impact analysis.

-- Prepare project plan.

- *A task.* Each one of the activities is decomposed into individual tasks needed to perform a specific activity.

For example, the *"Prepare project plan"* activity includes a *task* in which the effort estimates for the next phase are established.

1:M
The -------> notation means, a phase is decomposed into one or many segments, a segment is decomposed into one or many activities.

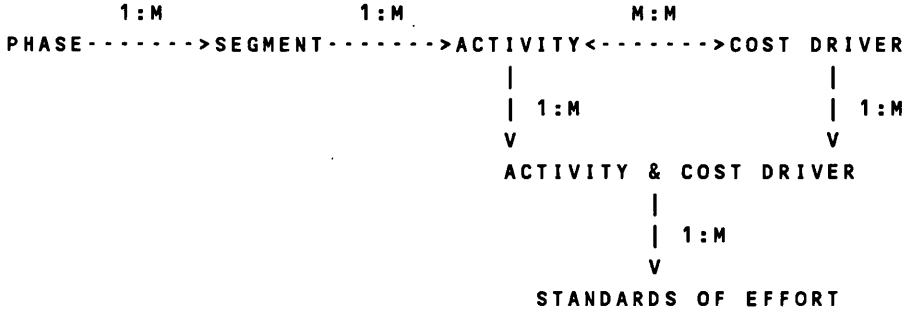
2. Cost drivers.

The model assumes that each activity is associated with a standard list of *Cost Drivers* involved in the process. The *Cost Drivers* serve as the basis for estimating tasks associated with each activity, during the estimation process. A cost driver is, for example, an input document, a report, a screen, a module, a contract , a proposed final software package, a request for proposal.

3. Standard of Effort.

A *Standard of Effort* is the (organizational) inverse of a standard rate of productivity, the amount of work required to accomplish one work unit. The standard of effort associated with each of the cost drivers and correlated activity, may differ, for an identical cost driver and associated activity, according to the **complexity** of the project. Some activities will vary in effort required depending on the number of users, potential availability of software, expected technical complexity, etc.

The *Effort Estimation Model (EEM)* is supported by a conceptual Software Development Life Cycle SDLC composed of phases, segments and activities , each activity is associated with one or more cost drivers which are correlated to standard of effort, as follows:-



M:M
The <------> notation means that each activity is associated with zero or many cost drivers, and each cost driver is associated with zero or many activities.

It is therefore clear that one cannot develop an *Effort Estimation Model* which is concerned with a particular phase, unless this phase is clearly described and understood by it users. This research addresses the effort estimation issue with regard to a single project.

Definition of the software life cycle phases.

Phase 1:- **Project Planning.**

Project Planning addresses the project initiation and feasibility issues. The **project definition** segment is part of this phase. It is often called the "Preliminary Survey". In this segment the preferred concept for the software project is defined, the software development strategies are developed and the superiority of the chosen concept to alternatives is presented. The Project Planning accepts the general needs or problems as inputs and produces a proposed comprehensive scope, an agreement on problems and a definition of a project which also includes a work plan for the next phase. The Project Planning phase is implemented by looking into the fact gathering and analysis, interviews and discussions, etc. This process usually lasts between **one week and one month.**

Phase 2:- **Preliminary System Design.**

The objectives of this phase are to determine how the new system (target system) should be implemented to meet the business needs of the organization and to obtain the commitment of the management to the proposed system, before the major portion of the project development cost is incurred. The main concerns of this phase lay in the following issues:-

- What the system does from the user's viewpoint?
- How does the system operate from a technical viewpoint?
- What are the estimated operating costs and benefits of the system.
- What are the estimated installation costs and time table.

This phase includes the *"specification requirements"* and *"product design."* It includes a complete, validated:-

- Functional and technical specification of the user's requirements.
- Design from the user point of view.
- Interfaces, performance, security and control requirements of the software product.
- A complete and verified specification of the:-
  - -- Overall hardware and software architecture.
  - -- Data models for the project.

Phase 3:- **Detailed Design and Implementation.**

The objectives of this phase are to finalize the system design and successfully install the system in the operational environment of the company. The phase includes the detailed design, code and debug, test and pre-operations. The objectives of this phase should be accomplished with:-

- Developed procedures.
- Trained users.
- Ensured acceptances of the system by both computer operation and user personnel.

Appendix A. includes the decomposition of the various software life cycle development strategies.

The products of the model.

Based on the answers given by users of the system, the data stored in the knowledgebase and the rules for manipulating them, the model would offer estimates for the manpower needed for each segment of the Preliminary System Design phase. By estimating the effort for the Preliminary System Design, the model will also indicate a **general approximation** of the effort needed for the Detailed Design and Implementation phase.

Upon receiving the estimates (in the later stages of this model development) the model will provide the project manager with advice on the following topics: an alternate schedule, milestones, required products at the end of each milestone, etc.

HOW DOES THE MODEL WORK.

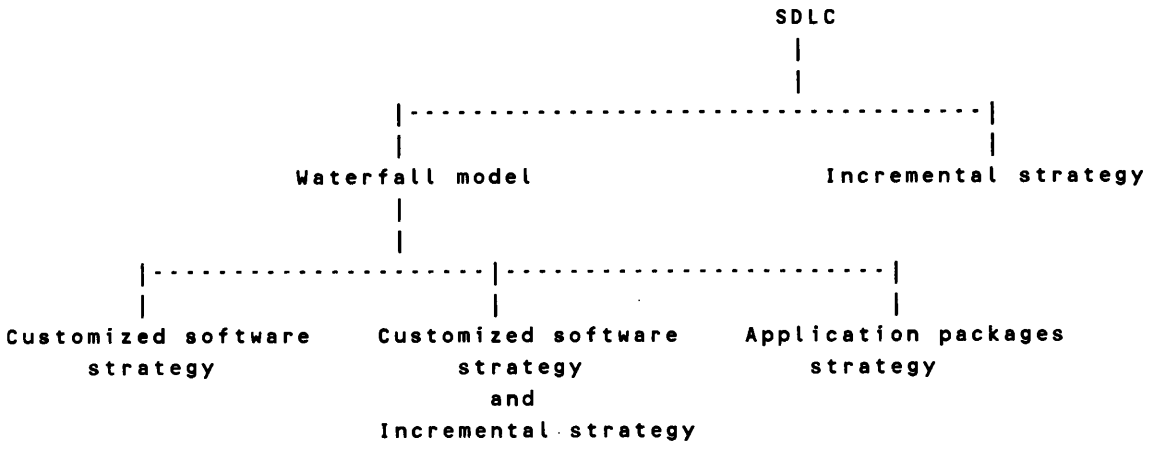
The model's sources of knowledge.

The model that supports the estimating process (which will use a combination of tools such as: expert system and decision support system tools, group decision processes, etc.), is based on two principlal sources of knowledge:-

- Basic knowledge:- A collection of data and rules that result from accumulated experience in various projects and organizations. Statistics regarding the productivity rate of the development process.
- Specific knowledge:- Produced by professionals involved in a given project which aims to assess its profile.

Project life cycle strategies.

The model recognizes three alternative strategies for the software development process. The first two strategies follow the conventional Waterfall model rationale [Boe76]. The Waterfall SDLC consists of discrete phases implemented in a defined sequence, each of which aims to achieve a defined set of sub-goals, before the next phase starts. These phases, also sequential, are very interdependent. Changes made in one phase have a significant influence on other phases. The third development strategy is geared toward a fluid environment with changing business needs as well as changing organizational infrastructure. This strategy provides a flexible enough route to allow different requirements to be defined and obsolete functions eliminated.



1. **Customized software development.**

This strategy follows the Waterfall SDLC model, however the approach does allow the integration of the new prototype paradigm into the classic flow of project development. The information gained from implementing a prototype, incorporating the traditional development procedures, can be added to the classic Waterfall paradigm and thus improve both the complex communication and the feasibility decisions involved in the process.

2. **Application packages system selection, design and installation.**

Application packages from many different industries are readily available and may save the developer much time because much of the time-consuming and costly ground work has already been done. One should not reinvent the wheel every time there is a need for a piece of software. This strategy also follows the Waterfall SDLC model. It is a sub-set of the Customized Software Development approach. When this strategy is taken, the SDLC is changed and includes some additional segments while others might be omitted.

3. **Incremental development.**

The new paradigm results from the new technological opportunities that have been developed since the introduction of the Waterfall model. The emergence of fourth generation languages and productivity tools for end-user computing has brought the need for a system development approach very different from the conventional one. This route varies distinctly from the Customized and Application packages strategies in that it allows for a "trial and error" approach to problem solving when a specific solution is impossible to pin-point initially.

When this strategy is chosen, the activity list decreases. However, there is no guarantee that the effort required for the entire system will be less than if implementing the same system using the custom-made approach, even so the time and cost of the first iteration may be substantially less.

The cycles of the model.

The software estimation process is an interactive and iterative process. It cannot produce fair estimates without consulting occasionally the estimator. The model uses different processes for the production of the estimation. The model works in the following cycles:

1. **The first cycle** includes the following:-

- Choosing the appropriate software development life cycle route.

- Assessing the characteristics and complexity of the project.

At the start of this cycle the estimator is asked which SDLC strategy is to be followed, as a result of strategy decisions. e.g. whether or not, software packages or an incremental approach are to be used. Whether the process of hardware selection is involved in the project or not, etc. Following this decision the estimator is asked to answer a series of questions on which the evaluation of the project characteristics such as size, complexity and risk, will be based.

The answers given by the estimator are based on the **general knowledge** available in the organization, even at this point of the project life cycle, as a result of the "preliminary survey".

2. **The second cycle** includes the following:

- Calculation of the required estimates, first iteration.

- Consultation: *Estimator - EEM.*

- Recalculation of the required estimates.

At the beginning of this cycle, the model tries to calculate the estimates. It may encounter difficulties in doing so with regard to a few activities characterized by a high variance of effort needed to accomplish them in different projects. For example, the "hardware and software selection" process will vary among organizations and types of projects. One cannot compare this process if done for a governmental agency with one done for a company in the private sector. The procedures vary, they might be much more complex in the public sector than in the private sector. One cannot compare the process of establishing criteria for selecting database management system (DBMS) software to be installed in a main frame computer that supports a distributed system to that of selecting DBMS software for a micro computer which aims to support the software development in the user's departments.

Wherever the model comes across activity of that sort, it consults the user and eventually offers an estimate.

If the estimator feels that the estimates do not reflect (accurately) his opinion, experience, intuition, etc., he informs the model, which responds by presenting all the questions and the answers he gave, for his reconsideration. The estimator is allowed to change them and the model will provide new estimates.

The end product of this cycle is the number of working days required to accomplish the activities included in each of the segments which compose the Preliminary System Design phase.

The required number of working days is derived from the estimator's answers, the knowledgebase of the model and the inference rules which manipulate both. The model will provide the estimator with the major assumptions that affect the estimates.

3. **The third cycle** includes the following:-

- Examining the principle components of the knowledgebase and making ad-hoc changes to them.
- Recalculating the required estimates.
- Approximating the required effort for the total project.
- Assessing project risk.

The chief concern of this cycle lies with fine tuning the major component of the model to the specific organization or project.

If at this time the estimator is still not satisfied with the estimates, the principal components of the knowledgebase are shown for his examination in the following two iterations:

- The **first iteration** covers the rules used by the model in determining the project complexity level. The estimator is asked to examine the rules included in the knowledgebase for the determination of the complexity level. He is authorized to change the rules that, in his opinion, are not suitable to this project environment.

- The **second iteration** covers the cost drivers which contribute to each of the activities and their correlated standard unit of effort. It is suggested that the estimator examine the:-

  -- Associations, proposed by the model, between the cost drivers and the activities.

  -- The *"standard of effort"* units attached to each of the integrated entities composed of cost factor and activity.

These components are also subject to change by the estimator.

P A R T 2 - THE QUESTIONNAIRE.

Instructions to the estimator.

1. Please, name the project to which you will relate your answers. Give a brief description of the scope of this project and its use.

_____

_____

_____

_____

_____

_____

_____

2. Please, read the questionnaire. **Do not answer it yet.** Examine the questions' relevance to the project under consideration and make sure you have with you the documentation which will enable you to answer them as precisely as possible. If not all of the documentation is available, **please try to answer all the questions even if you are not certain of all the answers. In such a case, base your information on your experience in similar projects or on your intuition.**

3. The questions are grouped into categories. You are asked either to select the appropriate answer or to indicate a quantitative value. If a question is not relevant to the project, please indicate so and move to the next question.

4. The instructions for each category of questions will introduce the group.

Please, try to answer even if your confidence in the answer is rather low

---

Remember, the system is designed to enhance the team, its considerations and judgements.

Please, feel free to comment, express your opinion or draw my attention to any question which may not be clear to you.

I am available for further information, explanation or facilitating the process of answering the questionnaire. Please, do not hesitate to contact me.

GROUP 1 - General background: project, organization and estimator.

I.0 Today's date_____

I.0.1 When had the project been developed:

Start date (month, year):_____

Finish date (month, year):_____

I.1 The name of the project:-

_____

1.2. The type of project. *
(Circle the appropriate option.)

* This estimation model is not appropriate for the development of support software.If your project is of this type please ask someone else in your organization to answer this questionnaire.

Application software.

1 Batch.
2 Interactive.
3 Batch and database
4 Interactive and database.
5 Scientific and engineering
6 Embedded or real time.
7 Multiple type application software.

8 *Utility software.*

9 Other. Please name:_____

I.3 Organization name:-

_____

I.3.1 Type of organization:
(Circle the appropriate option.)

1 Manufacturing industry.
2 Banking, insurance or financial institute.
3 Government / public service
4 Software house.
5 Retail.
6 Service industry. (non financial services)
7 Scientific / Engineering
8 Inter-organizational. (e.g. Swift)
9 Defence contractors.
10 Other. Please name:_____

I.3.2 Computer type.
(Circle the appropriate option.):-

1 Mainframe.
2 Mini.
3 Micro.
Please name._____

I.4 How was the required effort estimated?

1 Fully or partially automated estimating tool?
Please name:_____

_____

2 Formal manual estimating method.
Please name:_____

_____

3 Informal manual estimating method.
Please describe:_____

_____

I.5 Who estimated the required effort for the project?
(Circle the appropriate option.):-

1 Individual estimator.
2 Team members as individual estimators.
3 Team members as a group of estimators.

4. Other. Please name._____

_____

If the estimation process was done by a group of estimators, (your answer to question I.5 is 3), move to question number I.8 In this case, <u>each individual</u> estimator in the group estimation process should indicate his assignment in the developed project.

Estimator's backgroud:-

I.6 Surname:_____

First name:_____

I.7 Your title in the organization)

_____

I.8 **Your assignment in the developed project**
{Circle the appropriate answers} :-

1 Project leader.

2 System analyst.

3 System designer.

4 Programmer.

5 User

6 Quality assurance.

7 Technical assistant.

8 Data base administrator.

9 External consultant.

10 Professional (expert) estimator.

11 Other. If your assignment was defined differently, please name:

_____

_____

If the estimation process was done by a group of evaluators, ignore question I.9 and move directly to question I.10

I.9 **Describe briefly your background and experience.**
(Years as project leader, type of projects you managed or took part in, etc.)

_____

_____

_____

_____

The project development strategy.

I.10 **Which of the following software development strategies are suitable for the project under consideration? See page 6 for a detailed description.**

(Circle the appropriate answer)

1 Customized software development.

2 Customized software development and incremental development.

3 Application packages system development.

4 Incremental development.

5 Other. Please describe:_____

_____

P.1 **How many full time or part time team members were active at the Preliminary System Design phase of the project and in what skills?**

Remember, at this point of the project life cycle the estimates were implemented for the Preliminary System Design only. Therefore, only members that were active at that phase should be indicated here.

1 Project leader_____

2 System analyst_____

3 System designer_____

4 Programmer_____

5 User_____

6 Quality assurance_____

7 Technical assistant_____

8 Data base administrator._____

11 Other. Please name.

12 **Total team members:**_____

GROUP 2 - The profile and characteristics of the project.

The following group of questions aims to assess the profile and characteristics of the new system. The values I ask for are those you anticipated at the project planning phase. You are asked to indicate, for each question, three possible values: the highest, the lowest and the most likely estimated values, only if you recorded your estimations this way. If you did not record your estimations using a range of values use the "most likely " row to indicate your estimated values. You are also asked to indicate if this data was recorded for the project. If it was, please indicate it by circling the letter Y, if not circle the N. If the question is not relevant to the project, mark x and turn to the next question.

From my own experience, I realize that there are organizations that do not record regularly the information I am asking for. If your organization is one of these, please give the actual values, at the end of the project, for each of the following questions. Please indicate which is the case by circling the appropriate answer, I.11 or I.12.

I.11 The following are the estimated values at the Project Planning phase.

I.12 The following are the actual values at the end of the Project.

In case the project is composed of sub-systems, each sub-system should be estimated separately.

"PRESENT SYSTEM".

The "present system" is the one which was replaced by the project which is the subject of this questionnaire. **Remember,** a present system might also be a manual system.

P.2 Number of inputs (types, not volumes) in "present system"

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

P.3 Number of reports in "present system"

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

P.4 Number of screens in "present system"

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

Was this data recorded (P.2,P.3,P.4)? Y/N

TARGET SYSTEM (Replacement system)

Screen formatting

P.5 Number of screens.
(Each window is a screen)

1 *Inquiry screens.*

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

2 *Display screens.*

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

3 *Data entry screens.*

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

Report generating

P.6 Number of reports in the target system.

Only reports for users' application. Not including systems or error messages.

1 *Total number of reports expected:*

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

2 *Major reports:*

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

Batch data entry.

P.7 Number of batch inputs (types, not volumes) or input documents.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

On line data entry.

P.8 Number of on line inputs (types, not volumes) for updating.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

Resource requirement and capacity planning

High volumes For planning the resource requirements (capacity planning), in certain situations **where a high degree of estimating accuracy is necessary, or when response time is critical.**

P.9 Number of high volume inputs.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

P.10 Number of high volume inquiries.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

P.10.1 Number of high volume remote outputs transfer.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

P.11 Number of simulation models.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

Was this data recorded (P.9 - P.11)? Y/N

DESIGN FACTORS

P.12 Number of data elements in database divided by 100.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.13 Number of complex / major functions, from user design viewpoint.

The functions define the system's processing from the user's perspective. They are based on the business function. A complex function might be a function that involves data manipulation or statistical analysis such as:- calculating gross / net pay, calculating the re-order point, matching deliveries versus orders, application of cash e.g. billing, unique validation procedures, etc.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.14 Number of processes.

The processing functions are identified during the user design group of activities. Their identification provides a basis for identifying computerized processes, (e.g. group of logically related transactions, on-line processes, batch processes or internal processes) which are organized into programs and programming units. Processing one record type per work unit. (A type of record is for example: transaction, data-base segment, message, report line.)

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.15 Number of complex processes, from technical design viewpoint, but not directly related to input /output.

When a complex calculation or decision must be made while processing the data elements within a record.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.16 Number of modules, excluding control and utility modules.

A module is a discrete piece of work that has a definable product that can be tested. (e.g. validate a transaction, code to print a report).

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.17 Number of control and utility modules.

E.g. a module to manage file access and track updates, produce control reports, a database management control routine or a job control to run a periodical processing.
Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

OTHER FACTORS

P.18 Number of users, to be interviewed in order to determine the functional requirements of the system.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.19 Number of new or redesign forms.(e.g. output reports on preprinted forms).

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.20 Number of files to be converted.

Include also manual files which are to be computerized. (e.g. transaction, master or table look-up files, manual ledger).

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.21 Number of special file conversions.

When planning the conversion and / or the testing process of a new system, one should identify any work required on special facilities during the System Implementation phase, that are integral of the operational system, conversion system or testing procedures.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.22 Number of invitations to tender (requests for proposals *RFP*) to be developed, during the Preliminary System Design phase. (For example: hardware, application software, support software or subcontracting)

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.23 Number of tenders (requests for proposals) to be evaluated. (short list):

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.24 Number of contracts to be negotiated during the Preliminary System Design phase. For example: hardware, software, equipment maintenance, third party or internal contracts.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.25 How many software packages that were found suitable for in-depth evaluation, were actually evaluated at the Preliminary System Design phase?

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.26 Level of customization required. Number of modifications needed to fit the requirements.

1 Modifications were not required.

2 Only minor changes were required. Exclude extensive changes, such as updating programmes or file changes:

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

3 Extensive changes were required, such as updating programmes or file changes.

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____

Was this data recorded? Y/N

P.27 Was a working model or a prototyping approach used in this project to clarify the user's requirements?
Your answer is? **(Y/N)**
NO, move to the next group of questions.

YES, please describe your approach:-

_____

_____

P.28 What parts of the requirements was prototyped?

1 All I/O and functions were prototyped.
2 The majority of the inputs / outputs and the functions were prototyped.
3 Only a few of the I/O were prototyped?

P.29 What was the number of iterations for prototyping?

Mark x if the question is not relevant___
Most likely estimate _____
Low estimate _____
High estimate _____
Was this data recorded? Y/N

P.30 Were prototyping tools used in this process? Please name:_____

_____

GROUP 3. - Complexity classification and risk indicators

The following group of questions relates to a set of external factors which affect the system's complexity and thus the development process. Please, choose the appropriate answer and circle it. Some of the groupings may appear incompatible, but from the perspective of system complexity they are grouped naturally. For example: question G.5: "How committed is the upper level user management to the system?" has : "Somewhat reluctant, or unknown" as an optional answer. The two parts of the answer are not the same, however from a complexity view point it does not make any difference. This group of questions applies to the whole project *SDLC*.

P.31 Database environment.

How many logical databases, does the project incorporate?
A logical database is the application view of the database. (e.g. customers, claims, payments)

1 A single logical database
3 Two to four logical databases
5 Five or more logical databases

S.1 **What was the required (realistic) response time for high volume transactions (inquiry only)?**

1 Response time not a factor in this project.
2 Over 10 seconds.
3 5-10 seconds.
4 3-5 seconds.
5 1-3 seconds.

S.1.2 **System impact on financial status?**

1 Minor.
3 Major.
5 Critical.

S.1.3 **System impact on operational status?**

1 Minor.
3 Major.
5 Critical.

S.2 **Interface with other applications.**

1 Few interfaces
3 Several uni-directional interfaces.
5 Multi-directional interfaces with several applications.

S.2.1 **Number of departments (other then the IS) involved with the project?**

1 One
3 Two
5 Three or more

S.2.2 **Number of working units involved in the project? Was this project developed for the usage of one working unit or more?**

1 Single-site development for one working unit.
3 Single-site development for multiple working units within a single-site.
4 Single-site development for multiple working units within multiple-sites.
5 Multiple development sites or a multi-company project.

S.2.3 **If you propose to replace the system, what percentage of existing functions are replaced on a one-to-one basis?**

1 50% to 100%
3 25% to 50%
5 0% to 25%

S.2.4 **What is the severity of procedural changes in the user department caused by the proposed system?**

1 Low
3 Medium
5 High

S.2.5 Was this project a conversion or a functional repeat of a well known project?

1 Straight conversion. Code was transfered from one machine to another.
2 Functional repeat with some new features. (e.g. algorithm and logical design are well known).
3 An even mixture of repeated and new features.
4 A new system. Algorithm and logical design was developed from scratch.
5 A new system. Algorithm and logical design was developed from scratch. Hardware and software interfaces were defined as the design matured.

S.2.6 Does theuser organization have to change structurally to meet requirements of the new system?

0 No
1 Minimal
3 Somewhat
5 Major.

S.2.7 What is the general attitude of user?

1 Good - understands value of data processing solution.
3 Fair - sometimes reluctant
5 Poor - anti data processing solution.

<u>Technical environment.</u>

T.1 Computer type, operating system, installation aids and project familiarity.

1 Computer with simple operating system, good installation aids and project team is familiar with it.
2 Computer with average operating system, fair installation aids and all are generally familiar to project team.
3 (5) Computer with average operating system, fair installation aids and project team is unfamiliar with.
4 (5) Complex, large computer and operating system, includes installation aids which are generally familiar to the project team.
5 Complex, large computer and operating system which are not familiar to the project team.

T.1.2 Is any hardware new to the company?

0 None.
2 Most hardware is familiar to the project team.
3 (4) Most hardware is familiar to a part of the project team.
4 (5) Most hardware is new to the entire project team.
5 All hardware is new to the entire project team.

T.2 Data base management system.

1 Easy to use. Much prior experience.
3 Typical major DBMS with prior experience.
5 New or user customized.

T.3 On line monitor.

1 Easy to use. Much prior experience. Handles all system requirements.
3 Major monitor. No supplementary logic required.
5 New or user customized.

T.4 Data dictionary

1 Data dictionary with development aids is available.
3 Data dictionary without development aids is available.
5 New dictionary or no dictionary.

T.5 The development methodology

1 There is use of propriety structured methodology, for the whole life cycle.
3 There is no use of propriety structured methodology, but some usage of structured technique when designed.
4 (5) There is only limited use of structured techniques.
5 There is no use of structure techniques

T.6 Communication and distributed systems.

1 Established and there is no need for modification.
3 Established but some development required to support.
5 First time use.

T.7 System architecture.

1 Centralized. (single processor)
2 Coupled. (multiple processors)
3 Federated. (processors linked via bus)
4 Distributed. (centralized database)
5 Distributed. (distributed database)

Organizational environment.

G.1 Number of people whose working practice was affected by the system,

1 1-50 employees.
2 50-500 employees.
4 500- 2000 employees.
5 2000 or more employees.

G.1.1 Number of people whose working practice was affected by the system, in one working unit. If great variation in unit size, then give an average.

1 1-50 employees.
2 50-500 employees.
4 500- 2000 employees.
5 2000 or more employees.

G.2 User familiarity with the system. (How knowledgeable is the user representative in the proposed application area?)

1 Extensive, has been involved in prior implementation.
3 Considerable, understands the concept but no experience.
5 Minimal.

G.3 Decision makers

1 Key individuals.
3 Single committee with key individuals.
5 Multiple committees with multiple reviews.

G.4 Information processing service structure

1 Single decision maker.
3 Established, strong project management function.
5 Project has multiple decision makers, within complex organization

G.5 Commitment of the upper level user's management to the system?

1 Extremely enthusiastic.
3 Adequate.
5 Somewhat reluctant, or unknown.

Project team composition

C.1 Project team structure

1 Single decision maker and less than four team members.
2 Four to eight project team members. Some technical assistance is required.
3 (5) Large project team and **multiple** decision makers.
4 (5) Large team and **matrix organization.**
5 Ambiguous or uncertain project organization structure.

C.2 Experience with industry / application

1 Extensive, high degree of capability.
3 Considerable, previous exposure but limited knowledge.
5 Minimal, first exposure.

C.3 Technical experience

1 Extensive, all required systems software has been successfully used before.
3 Considerable, small learning curve anticipated.
5 None, major learning curve.

C.4 Staffing and Hiring.

1 There was no need to hire new personnel for this project.
3 Personnel could be hired as justified, for this project.
5 There is a need to hire personnel for this project but hiring is difficult.

GROUP 4. Actual effort data.

This part of the questionnaire aims to capture historical data, related to the actual and original estimates. In this part you are asked to relate to the various project life cycle phases (SDLC). Therefore, to remind you, this research uses the following life cycle phases:

- Project planning. This phase addresses the project initiation and feasibility issues. The preferred concept for the software project is defined, the software development strategies are developed and the superiority of the chosen concept to alternatives is presented.

- Preliminary System Design. This phase is the start-up point of the project, in which the software product is specified. This phase includes the *"specification requirements"* and the *"product design specification"*. It includes a complete, validated:-

- Functional and technical specification of the user's requirements.
- Design from the user point of view.
- Interfaces, performance, security and control requirements of the software product.
- A complete and verified specification of the:-
  - -- Overall hardware and software architecture.
  - -- Data models for the project.

- Detailed Design and Implementation. This phase includes the *"detailed design", "code and debug", "tests, system integration and pre-operations"*.

Please, follow the project development strategy *(SDLC)* you had chosen (your answer to question I.0) in appendix A. You are asked to cancel the segments and/or activities that you did not follow in this project and to add those you did follow but are not included in my list of activities.

A.1. What was the total actual effort required in this project?
Total person hours:-_____

A.1.2 What was the actual effort required in this project for the Preliminary System Design phase?
Person hours:-_____

A.1.3 If your project development life cycle (SDLC) phases do not agree with those given in this research, please group the following categories so that they correspond with yours:

Percentage of effort

Preliminary System Design effort.
Total person hours:-_____%

Detailed Design and Implementation effort.
Total person hours:-_____%

A.2 What was the elapsed time for the total project?
Weeks:-_____

A.2.1 What was the elapsed time for the Preliminary System Design?
Weeks:-_____

A.2.3 If your project development life cycle (SDLC) phases do not agree with those given in this research, please group the following categories so that they correspond with yours:

Percentage of elapsed time

Preliminary System Design elapsed time.
Total weeks:-_____ %

Detailed Design and Implementation elapsed time.
Total weeks:-_____ %

A.3 How many full time persons worked in the total project?
Number of full time persons:_____

A.3.1 How many of full time persons worked in the Preliminary System Design.
Number of full time persons:-_____

A.3.2 Does your actual effort include the user's departments effort?
Your answer is ?(Y/N)
No, move to question number A.4

Yes, What percentage of the effort was due to the users' departments?

Throughout the Preliminary System Design:_____%.

Throughout the Detailed Design and Implementation:-_____%

A.4 What was the peak time in the project? How many weeks from the starting date of the project was it staffed the most?
Weeks:-_____

A.5 What was the total number of source lines of code in the project?
Do not include comments.Include new, modified and unmodified code.
Total lines of code:-_____

A.6 What was the major language used in the project?_____

What percentage of the work was done using this language?_____%

A.6.1 What was the secondary language used in the project?_____

What percentage of the work was done using this language?_____%

A.7 Did you use fourth generation languages?
Your answer is ?(Y/N)
No, move to question A.8

Yes, which languages?_____

What percentage of the work was done using this language?

Throughout the Preliminary System Design:-_____%

Throughout Detailed Design and Implementation:-_____%

A.8 Did you use screen formatting tools?
Your answer is ?(Y/N)
No, move to question number A.9

Yes, which tools and what percentage of the work was done using them?
Please, name the tools:-_____

_____

Throughout the Preliminary System Design:-_____%

Throughout Detailed Design and Implementation:-_____%

A.9 Did you use report generator tools?
Your answer is?(Y/N)
No, move to question number A.8

Yes, which tools and what percentage of the work was done using these tools?
Please, name the tool:_____

_____

Throughout the Preliminary System Design:-_____%

Throughout the Detailed Design and Implementation:-_____%

A.10 Was this project, in your opinion, of complex, moderate or simple difficulty?

1 Simple.
2 Moderate.
3 Complex.

The following group of questions addresses the activities which vary between projects and organizations.

E.1 Did this project include the design for complex processes such as new authorization or control?

Your answer is ?(Y/N)
No, move to question number E.2

Yes, what were these processes? (describe)

_____

_____

_____

E.1.1 What, was the effort needed to accomplish this activity in the project?
Work hours_____

E.2 Did the project include the design of control and utility modules?
Your answer is ?(Y/N)
No, move to question number E.4

Yes, what were these utilities? (Please name):

_____

_____

E.2.1 What, was the effort needed to accomplish this activity in the project?
Work hours_____

E.4 Did the project include the design of complex functions?

Based upon the business functions included in the scope of the project, the computerized processing functions are defined. For example, the associated computerized processing function for the "order entry business function" may be an order entry on line conversation. Such conversation may take the form of:-

- Identify customer.
- Enter order.
- Enter line header.
- Recapitulate order.

But, in more complex applications each of the above items could be a conversation by itself.
Your answer is? (Y/N)
No, move to question E.5.

Yes, move to the next question.

E.4.1 What, was the effort needed to accomplish this activity in your project?
Work hours:-_____

Describe the complex functions:-

_____

_____

_____

E.5 Did this project include the design for communication and distributed systems?
Your answer is ?(Y/N)
No, move to question number E.6

Yes. (Circle the appropriate answer)

1 Communication systems only.
2 Distributed systems only.
3 Both.

E.5.1 What was the effort needed to accomplish this activity in your project?
Work hours:-_____

E.6 Was hardware and software selection part of the project?
Your answer is? (Y/N)
No, move to question E.7.

Yes. (Circle the appropriate answer)

1 Hardware only.
2 Software only.
3 Both.

E.6.1 What was the effort needed to accomplish this activity in your project?
Work hours:-_____

E.7 Was special file conversion part of the project?
Your answer is? (Y/N)
No, move to group 4.

E.7.1 What was the effort needed to accomplish this activity in your project?
Work hours:-_____

"A CONFESSION":

A.12 What were your estimates for the required person hours at the Project Planning phase?

1 Estimated person hours for the total project:-

Person hours:-_____

Were these estimates recorded? (Y/N)

2 Estimated person hours for the Preliminary System Design phase:-

Person hours:-_____

Were these estimates recorded? (Y/N)

3 If your project SDLC phases do not agree with those given in this research, please group the following categories so that they correspond with yours:

Percentage of estimated effort

Preliminary System Design effort.
Total person hours:-_____%

Detailed Design and Implementation effort:
Total person hours:-_____%

A.13 What were your estimates for the required elapsed time (in weeks) at the Project Planning phase?

1 Estimated elapsed weeks for the total project:-

Weeks:-_____

Were these estimates recorded? (Y/N)

2 Estimated elapsed weeks for the Preliminary System Design:-

Weeks:-_____

Were these estimates recorded? (Y/N)

3 If your project SDLC phases do not agree with those given in this research, please group the following categories so that they correspond with yours:

Percentage of elapsed time

Preliminary System Design elapsed time.
Total weeks:-_____ %

Detailed design and Implementation elapsed time.
Total weeks:-_____%

A.14 What were your estimates for the required full time persons at the Project Planning phase?

1 Estimated full time persons for the total project:-

Number of full time persons:-_____

Were these estimates recorded? (Y/N)

2 Estimated full time persons for the Preliminary System Design:-

Number of full time persons:-_____

Were these estimates recorded? (Y/N)

3 If your project SDLC phases do not agree with those given in this research, please group the following categories so that they correspond with yours:

Percentage of full time persons

Preliminary System Design.
Full time persons:-_____%

Detailed design and Implementation.
Full time persons:-_____%

A word of thanks

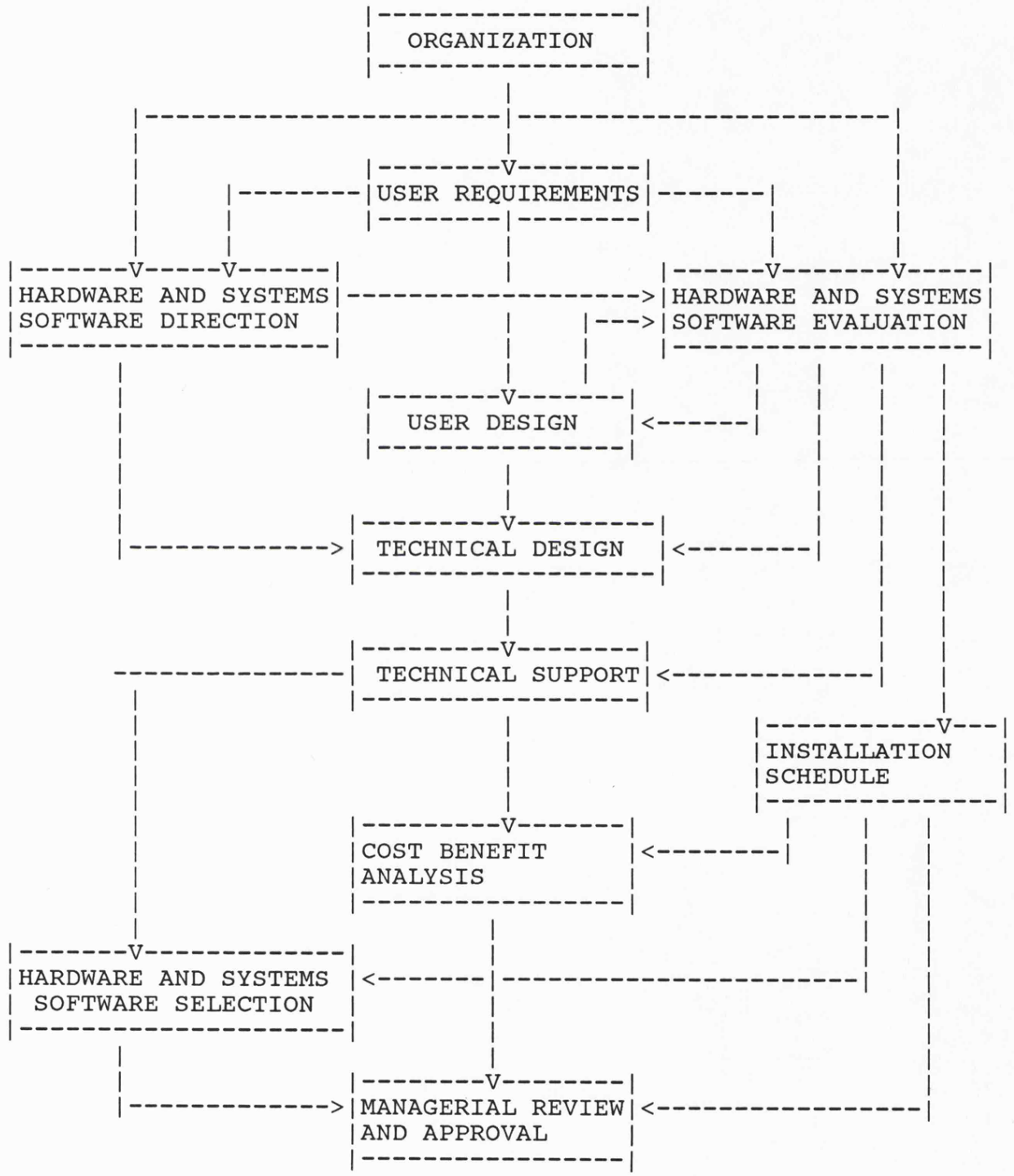
Now that you have survived to this point, thank you very much for your perseverance in the task

Zeeva Levy

Appendix A. Decomposition of the various *SDLC* strategies.

Appendix A.1:- Decomposition of the Preliminary System Design Phase for the Customized Software Development strategy.

The Preliminary System Design phase, if the Customized Software development strategy, has been applied, is composed of the following segments:-



The following activities composite of these segments:

Organization
Organize project

Hardware and systems software direction
Review company policies related to H&S.
Identify and evaluate hardware alternatives.
Identify and evaluate software alternatives.
Determine overall evaluation criteria *.
Evaluate alternatives
(Only for final proposal.)
Establish hardware approach.
Establish software approach.

Hardware and system software evaluation
Identify modification to system software.
Write direction for technical and functional specification.
Obtain requirements inventories for proposed application system's function.
Evaluate packages against the systems requirement criteria.
Summarize evaluation of each package for comparison.
Identify and evaluate hardware alterntives.

User requirements
Review present system
Identify functional requirement
Identify other requirements (Such as performance and security and control requirements)

User design
Initiate prototype [define scope, etc.']
Define inputs and outputs
Define processing functions *
Define data requirements
Issue preliminary functional specifications.

Technical design
Design technical architecture. *
Design data base .
Design system processes *
Design other processes *

Technical support
Design testing and conversion processes.
Determine resource requirements

Installation schedule
Identify installation steps
Establish personnel requirements
Develop conversion approach
Develop installation work plan

Note: An iterative process will take place at that point.

Cost/benefits analysis
Estimate installation costs.
Estimate operating costs and benefits.
Document the intangibles
Summarize overall economics

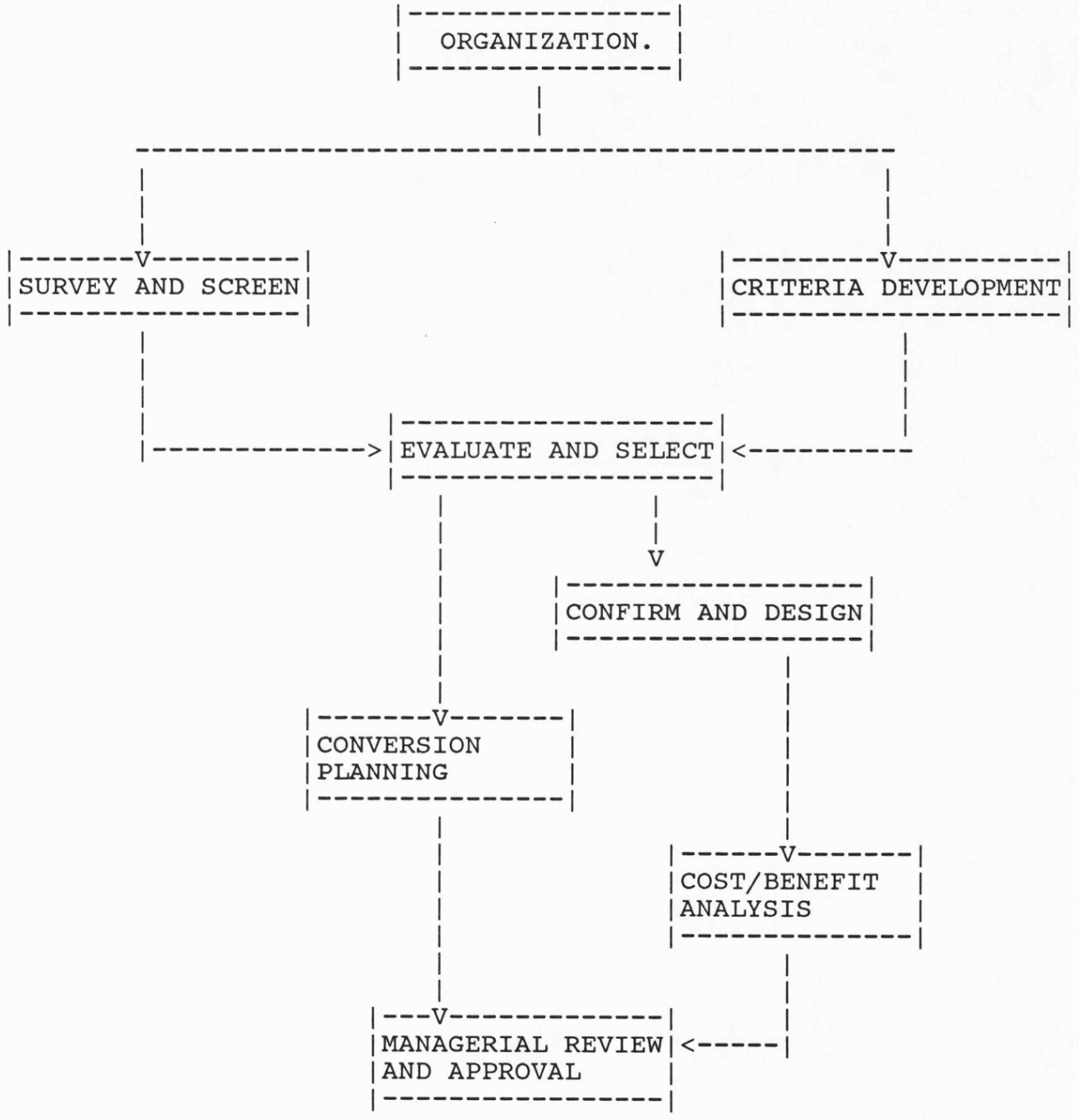
Hardware and software selection
Finalize selection
Negotiate contract terms

Management review and approval
Publish specifications.
Review with management.
Prepare management report.
Approve project and priority.

* These activities are characterized by high variance among projects and organizations

Appendix A.2:- Decomposition of the Preliminary System Design for the Application Packages Strategy.

If the Application Software Development Strategy, has been applied, the Preliminary System Design and selection phase, is composed of the following segments:-



The following activities composite of these segments:-

Organization
Organize project.
Set scope.

Survey and screen.
Survey packages.
Survey requirements.
Develop preliminary criteria.
Select finalists.

Criteria development.
Review present system.
Identify functional requirement.
Identify other requirements (Such as performance and security and control requirements, reliability requirements, response requirements, real time requirement, interactive requirements, etc.)
Define functional criteria.
Identify technical architecture.
Evaluate technical alternatives.
Define technical criteria.
Finalize selection criteria.

Evaluate and select.
Develop contract strategy.
Obtain information and training.
Evaluate features.
Compare costs and benefits.
Make preliminary selection
Prepare report.
Negotiate contract terms.
Obtain approval to continue.

Confirm and design.
Define application flow.
Prepare for acceptance test.
Define modification approach.
Perform acceptance test.
Design interfaces and modifications.
Define testing approach.
Define resource requirements.

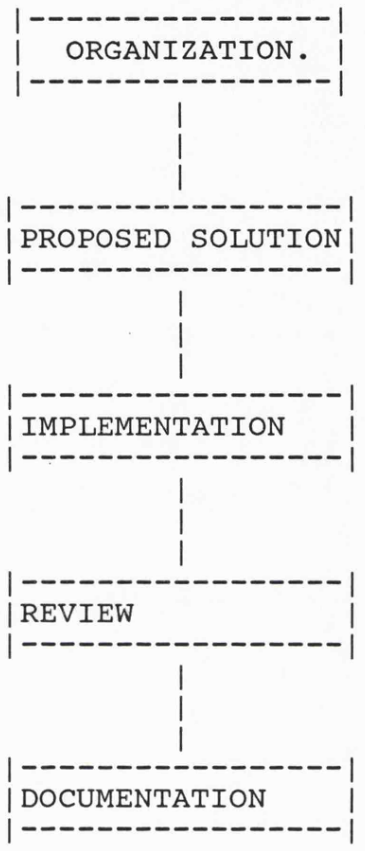
Conversion planning
Define conversion approach.
Define conversion processes.
Define conversion files.
Define conversion resource requirements.
Develop implementation plan.
Establish personnel requirements.

Cost benefit analysis.
Estimate implementation cost.
Estimate operation costs and benefits.
Document intangibles.
Summarize overall economics.

Management review and approval.
Publish specifications.
Prepare management report
Review with management.
Approve project and priority.

Appendix A.3:- Decomposition of the SDLC for the Incremental Strategy.

If the Incremental Development Strategy, has been applied, the SDLC is composed of the following segments:-



The following activities composite of these segments:-

Organization
Set scope
Define the problem

Proposed solution
Define inputs and outputs
Design logical data.
Select tools /techniques.
Define iterative strategy
Define technical architecture

Implementation.
Design Processing logic.
Design physical data
Implement processing logic
Test implementation
Reconfirm scope

Review
Establish next cycle

Documentation
Develop user documentation
Finalize system documentation

Appendix 7A. Decomposition of the Preliminary System Design Phase

The Preliminary System Design Phase is composed of the following segments:

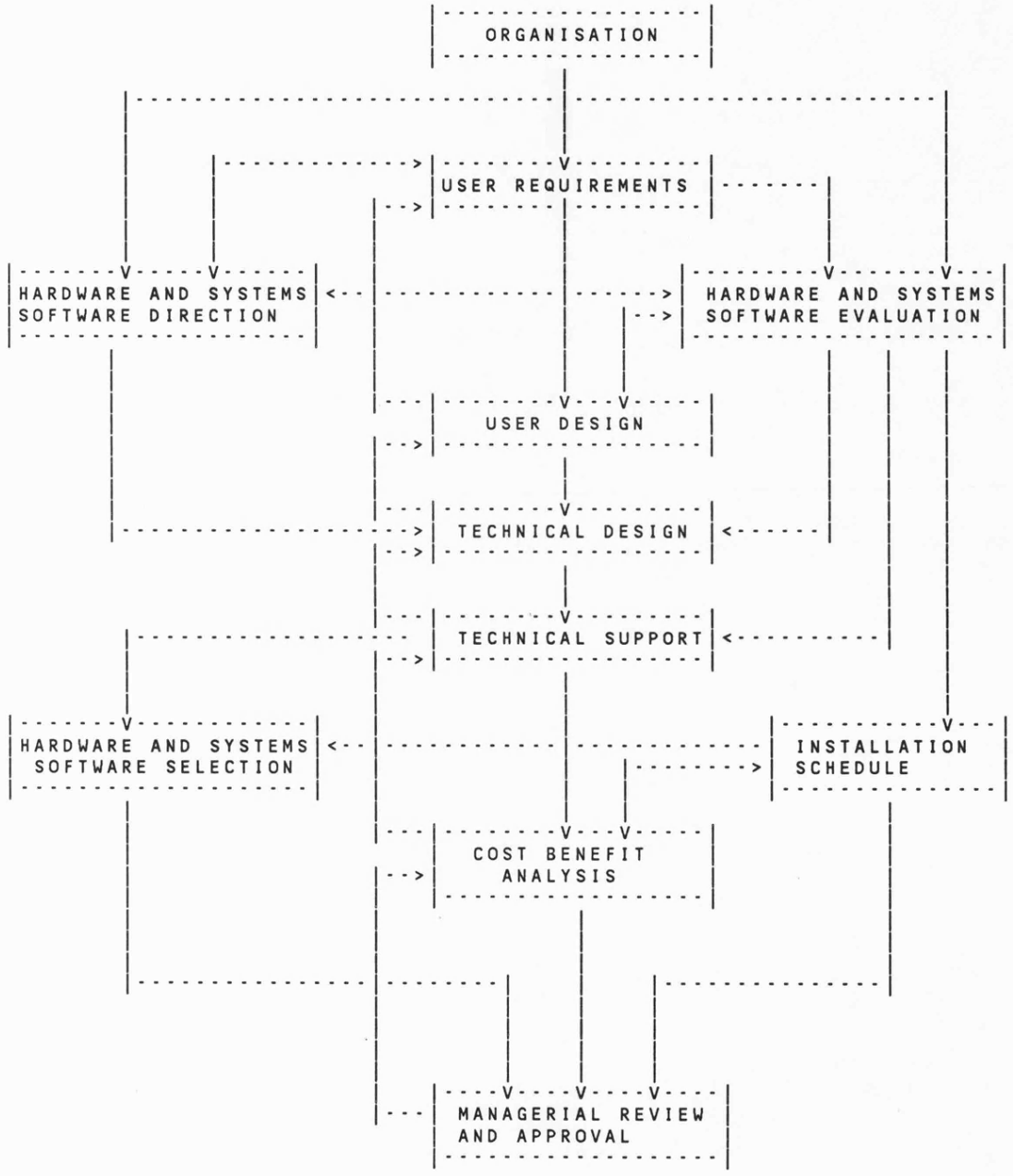


Figure 7.11 Decomposition of the PSD Phase into segments.

The following activities encompass these segments:

Organisation
Organise project

Hardware and systems software direction
Review company policies related to H&S.
Identify and evaluate hardware alternatives.
Identify and evaluate software alternatives.
Determine overall evaluation criteria *.
Evaluate alternatives
(Only for final proposal.)
Establish hardware approach.
Establish software approach.

Hardware and system software evaluation
Identify modification to system software.
Write direction for technical and functional specification.
Obtain requirements inventories for proposed application system's function.
Evaluate packages against the systems requirement criteria.
Summarise evaluation of each package for comparison.
Identify and evaluate hardware alternatives.

User requirements
Review present system
Identify functional requirement
Identify other requirements (Such as performance and security and control requirements)

User design
Initiate prototype [define scope, etc.']
Define inputs and outputs
Define processing functions *
Define data requirements
Issue preliminary functional specifications.

Technical design
Design technical architecture. *
Design data-base .
Design system processes *
Design other processes *

Technical support
Design testing and conversion processes.
Determine resource requirements

Installation schedule
Identify installation steps
Establish personnel requirements
Develop conversion approach
Develop installation work plan

Note: An iterative process will take place at that point.

Cost/benefits analysis
Estimate installation costs.
Estimate operating costs and benefits.
Document the intangibles
Summarise overall economics

Hardware and software selection
Finalise selection
Negotiate contract terms

Management review and approval
Publish specifications.
Review with management.
Prepare management report.
Approve project and priority.

* These activities are characterised by high variance among projects and organisations

Appendix 7A.1 Decomposition of the SDLC for the Iterative Strategy.

If the Iterative development strategy is applied, the SDLC is composed of the following segments:

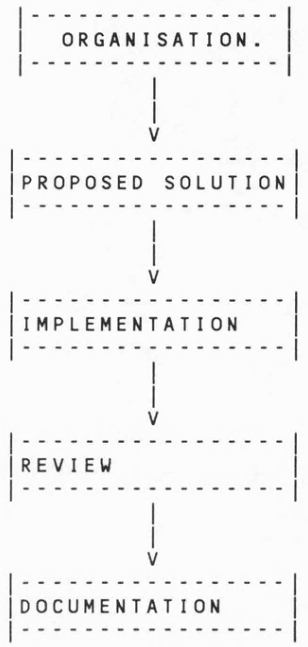


Figure 7.12 Decomposition of the iterative approach into segments.

The following activities composite of these segments:

Organisation
Set scope
Define the problem

Proposed solution
Define inputs and outputs
Design logical data.
Select tools /techniques.
Define iterative strategy
Define technical architecture

Implementation.
Design Processing logic.
Design physical data
Implement processing logic
Test implementation
Reconfirm scope

Review
Establish next cycle

Documentation
Develop user documentation
Finalise system documentation

Appendix 7B List of cost drivers used in the EEM.

Number of new systems (equals 1, each subsystem is estimated separately)

Number of project team members

Number of interviews.

Number of old reports, screens, inputs.

Number of new reports

Number of new screens: inputs,outputs and massages.

Number of new inquiries.

Number of new or re-design forms.

Number of data elements in data-base divided by 100.

Number of files conversion processes.

Number of invitations to tender (requests for proposal - RFP)
to be developed, during the PSD.

Number of requests for proposal to be evaluate, short list.

Number of contracts to be negotiated during the PSD.

Number of high volume inputs and or inquiries.

Number of simulation models.

Number of (working models?) or iterations for prototyping?

Number of application packages to evaluate (short list)

Number of modification required in software application package to fit
the requirement

* Number of complex functions.

* Number of complex processes.

* Number of utility and control modules, only.

* Number of special processes or files to be converted.

* The EEM collects the following information: description of the functions implemented and the actual effort on the following two activities:

Hardware and software selection

Communication and distributed systems

Appendix 7C. The EEM structure and concepts - examples

Examples are given to clarify the concepts described and implemented by the EEM. Since the EEM estimates the effort required for the PSD, all the segments and activities indicated in this example are of the PSD. The selection of the activities and cost drivers in the examples aims to introduce gradually the concepts used by the EEM and to cover the variety of its usage. The examples show that the 'standard of effort', the resources consumed by a cost driver, may differ in the implementation of various activities. Two types of cost drivers are introduced, cost drivers which are size attributes of the product and cost drivers which are identified as an overhead resources.

The primary aim of the 'user requirements' **segment** is to define the functional requirements of the system that support the business needs of the users. It is in this segment that the information flows (manual or computerised) of the current system are determined, the business functions to be implemented in the target system are described in detail and additional performance requirements are specified. Therefore, in aiming to identify the 'user requirements', we need to implement the following **activities:**

* Review the present system
* Identify the functional requirements
* Identify requirements such as performance, security and control

The 'user design' segment aims to transform the functional requirements defined in the 'user requirements' segment into a design from the user viewpoint, in business-oriented terminology. In this segment, the user interaction with the system, the inputs and outputs, and the business data managed by the system are defined in detail. Thus, when implementing the 'user design' segment we need to:

* Define inputs and outputs
* Define data requirements

Attempting to 'review the present system' we need to understand how it functions. This understanding is gained, in part, by interviewing key people in the

users' organisation and by studying the reports and enquiries, currently provided. Therefore, the number of interviews to be held for reviewing the current system and the number of reports, screens or inputs supplied by the present system are major contributors to the cost associated with this activity and, thus, with the 'user requirements' segment. See Table 7C.1 for an example of activities and their associated cost drivers.

Segments / Activities	The cost drivers associated with an activity	
Organisation		
Organise project	Number of project team members*	
User requirements		
Review present system	Number of interviews Number of reports, screens and inputs in the replaced system.	
Identify functional requirement	Number of new reports Number of new screens, inputs and outputs	>>> ***
Identify other requirements	Number of new systems*	
User design		
Define inputs and outputs	Number of new reports Number of new screen inputs and outputs Number of new inquiries Number of new or re-designed forms.	>>> ***
Define data requirements	Number of data elements in data-base	^^^
Technical design		
Design data-base	Number of data elements in data-base	^^^

* This is an example of a cost driver which is identified as an overhead resource.

>>>; ***; ^^^ These indicate cost drivers which are common to more than one activity in this example.

Table 7C.1 Examples of activities and their associated cost drivers.

Identifying the functional requirements of the target system involves understanding the reports, screens, inputs and outputs that are needed. Hence, the number of new reports, new screens, new inputs and outputs are major factors contributing to the effort involved in implementing this activity and, thus to the 'user requirements' segment. This understanding is also required for the definition

of inputs and outputs. However, the effort consumed in the understanding of these articles, in the context of the 'user design' segment, differs from the effort required for gaining this understanding when identifying the 'user requirements'. Hence, the 'standard of effort' required for an identical cost driver will vary when implemented in conjunction with various activities. In addition, when defining the inputs and outputs of the target system we might identify requirements for inquiries and for the re-design of forms. Hence, there are also cost drivers associated with this activity and thus, with the 'user design' segment.

The major contributor to the definition of the data requirements and to the design of the data-base is the number of data elements anticipated in the data-base. The number of data elements is identified as a cost driver associated with the 'define inputs and outputs' and 'design data-base' activities and therefore, contributes to the effort required for both the 'user design' and the 'technical design' segments. Nevertheless, the resources consumed by this cost driver in the implementation of these two activities differ. Hence, different 'standards of effort' are associated with this cost driver in various activities.

The activity 'identify other requirements' has a cost driver which represents the overhead effort required for a system. However, additional effort might be needed to implement this activity when high performance plays an important role in the target system, or when specific security and control features are required. It is obvious that a 'standard of effort' cannot be established for implementing unique requirements, since the effort consumed will differ widely among environments. Hence, the estimated effort for this activity is the result of the estimator's judgement (apart from the system overhead). The EEM collects performance data of such activities in various projects. This data will be used in estimating effort for projects where similar functions are required within similar environments.

The 'standard of effort' associated with each activity and cost driver is a function of the general complexity of the system. Therefore, a set of 3 values for the 'standard of effort' is correlated with each of the combined entities (activity and associated cost driver). Each value represents a different level of complexity. As previously discussed, and it is worth re-emphasising, the 'standard of effort' is a result of measurement of projects histories in heterogeneous environments, they were fine-tuned throughout this research. The concept is shown in Table 7C.2.

Segments / Activities	Standard of Effort associated with the activities and cost drivers			
Organisation				
Organise project	Number of project team members			
	SOE	10 *	if system is Simple	(S)
		20	if system is Moderate	(M)
		30	if system is Complex	(C)
User requirements				
Review present system	Number of interviews			
	SOE	3	if system is Simple	
		4	if system is Moderate	
		6	if system is Complex	
	Number of reports, screens and inputs in the old system			
	SOE	1	if system is Simple	
		2	if system is M or C.	
Identify functional requirement	Number of new reports			>>>
	SOE	2	if system is S. or M. or C.	***
	Number of new screens: inputs, outputs			
	SOE	2	if system is S. or M. or C.	
Identify other requirements	Number of new systems			
	Estimator's judgement for unique requirements.			
User design				
Define inputs and outputs	Number of new reports			>>>
	Number of new screens, inputs and outputs			***
	SOE	3	if system is S. or M. or C.	
	Number of new enquires			
	Number of new or re-designed forms.			
	SOE	4	if system is S. or M. or C.	
	for the cost drivers: new enquiries & re-designed forms			^^^
Define data requirements	Number of data elements in data-base			
	SOE	10	if system is S. or M. or C.	
Technical design				
Design technical architecture	Number of new systems			
	SOE	40	if system is Simple.	
		60	if system is Moderate.	
		180	if system is Complex.	^^^
Design data-base	Number of data elements in data-base			
	SOE	20 if system is Moderate.		

* This should be read 10 Person Hours is the resources (SOE) consumed for each team members to support the activity 'organise the project', when the system is of simple complexity.

Table 7C.2 An example of 'standard of effort' associated with an activity and a cost driver.

Appendix 7D. Complexity and risk determinants and rules for calculation.

7D.1 COMPLEXITY AND RISK CALCULATION

The complexity determinants are phrased as questions with options for answers which are scored between 0 and 5, from which the estimator is asked to choose the appropriate answer. These scores are totalled for each category of complexity (as previously discussed) and the added score is assessed as simple, moderate or complex according to the range it falls in. When the number of determinants included in each category is multiplied by 1, it classifies the lower level for 'moderate' complexity and when multiplied by 3, minus 1, it indicates the lower level of 'complex' complexity. Since 14 determinants encompass the General System Complexity, the ranges are classified as follows:

* Simple 0 - 13
* Moderate 14 - 40
* Complex 41 - 70

7D.2 ASSESSMENT OF GENERAL COMPLEXITY

The EEM uses set of rules to assess the system's complexity and the project risk, they are given below. Two groups of attributes are considered in the assessment of the 'general system complexity', they are size attributes of the target system and environmental system characteristics.

Group A: Attributes of product size

P.5* **Number of inquiry screens.** (Each window is a screen)

The complexity rule is: <u>number of inquiry screens</u> less than 10 is scored 1

greater or equal than 10 and less than 20 is scored 2

greater or equal than 20 is scored 3

* The code adjacent to each determinants refers to the question numbers used in the questionnaire.

P.6 **Number of reports in the target system.**

(Only reports for users' application. Not including systems or error messages.)

P.6.1 *Total number of reports expected:*

(P.6.1 itself is not a complexity determinant)

P.6.2 *Major reports:*

The complexity rule is: <u>number of major reports</u> less than 10 is scored 1

greater or equal than 10 and less than 30 is scored 2

greater or equal than 30 is scored 3

Minor reports: (P.6.2-P.6.1)

The complexity rule is: <u>number of minor reports</u> less than 20 is scored 1

greater or equal than 20 and less than 40 is scored 2

greater or equal than 40 is scored 3

<u>Batch data entry</u>

P.7 **Number of batch inputs (types, not volumes) or input documents.**

The complexity rule is: <u>number of batch inputs</u> less than 10 is scored 1

greater or equal than 10 and less than 30 is scored 2

greater or equal than 30 is scored 3

<u>On line data entry</u>

P.8 **Number of on-line inputs (types, not volumes) for updating.**

The complexity rule is: <u>number of on-line data entry</u> less than 5 is scored 1

greater or equal than 5 and less than 20 is scored 2

greater or equal than 20 is scored 3

P.12 **Number of data elements in data-base divided by 100.**

The complexity rule is: <u>number of data elements in DB</u> less than 200 is scored 1

greater or equal than 200 and less than 500 is scored 2

greater or equal than 500 is scored 3

P.13 Number of complex / major functions, from user design viewpoint.

The complexity rule is: number of complex functions less than 30 is scored 1

greater or equal than 30 and less than 50 is scored 2

greater or equal than 50 is scored 3

GROUP B: Environmental systems complexity

P.27.1 Data-base environment.

How many logical data-bases, does the project incorporate?

A logical data-base is the application view of the data-base. (e.g. customers, claims, payments).

1** A single logical data-base.

3 Two to four logical data-bases.

5 Five or more logical data-bases.

**The numbers in the first column are the scores.

S.1 What was the required (realistic) response time for high volume transactions (inquiry only)?

1 Response time not a factor in this project.

2 Over 10 seconds.

3 5-10 seconds.

4 3-5 seconds.

5 1-3 seconds.

S.1.2 System impact on financial status?

1 Minor.

3 Major.

5 Critical.

S.1.3 System impact on operational status?

1 Minor.

3 Major.

5 Critical.

S.2 Interface with other applications.

1 Few interfaces.

3 Several unidirectional interfaces.

5 Multi-directional interfaces with several applications.

S.2.3 If you propose to replace the system, what percentage of existing functions are replaced on a one-to-one basis?

1 50% to 100%.

3 25% to 50%.

5 0% to 25%.

G.2 User familiarity with the system. (How knowledgeable is the user representative in the proposed application area?)

1 Extensive, has been involved in prior implementation.

3 Considerable, understands the concept but no experience.

5 Minimal.

Application software evaluation

P.25 How many software packages that were found suitable for in-depth evaluation, were actually evaluated at the Preliminary System Design Phase?

Few	1-2	simple.
Several	3-5	moderate.
Major	5+	complex.

P.26 Level of customisation required. Number of modifications needed to fit the requirements.

1 Modifications were not required: - **No effect (0).**

2 Only minor changes were required. Exclude extensive changes, such as updating programmes or file changes: **Moderate (3).**

3 Extensive changes were required, such as updating programmes or file changes. **Complex (5).**

7D.3 ORGANISATIONAL ENVIRONMENT

S.2.1 Number of departments (other then the IS) involved with the project?

1 One.

3 Two.

5 Three or more.

S.2.2 Number of working units involved in the project? Was this project developed for the usage of one working unit or more?

1 Single-site development for one working unit.

3 Single-site development for multiple working units within a single-site.

4 Single-site development for multiple working units within multiple-sites.

5 Multiple development sites or a multi-company project.

S.2.4 What is the severity of procedural changes in the user department caused by the proposed system ?

1 Low.

3 Medium.

5 High.

S.2.5 Was this project a conversion or a functional repeat of a well known project?

1 Straight conversion. Code was transferred from one machine to another.

2 Functional repeat with some new features, e.g. algorithm and logical design are well known.

3 An even mixture of repeated and new features.

4 A new system. Algorithm and logical design were developed from scratch.

5 A new system. Algorithm and logical design were developed from scratch. Hardware and software interfaces were defined as the design matured.

S.2.6 Does user organisation have to change structurally to meet requirements of the new system ?

0 No.

1 Minimal.

3 Somewhat.

5 Major.

The following factors should affect mainly user oriented segments such as user requirements, user design, system test.

S.2.7 What is the general attitude of user?

1 Good - understands value of data processing solution.

3 Fair - sometimes reluctant.

5 Poor - anti data processing solution.

G.1 Number of people whose working practice was affected by the system.

1 1-50 employees.

2 50-500 employees.

4 500- 2000 employees.

5 2000 or more employees.

G.1.1 Number of people whose working practice was affected by the system, in one working unit.

If great variation in unit size, then give an average.

1 1-50 employees.

2 50-500 employees.

4 500- 2000 employees.

5 2000 or more employees.

Only the highest score between G.1 and G.1.1 determinants is used to calculate the complexity indicator for the 'technical environment'.

G.3 Decision makers.

1 Key individuals.

3 Single committee with key individuals.

5 Multiple committees with multiple reviews.

G.4 Information processing service structure.

1 Single decision maker.

3 Established, strong project management function.

5 Project has multiple decision makers, within complex organisation.

G.5 Commitment of the upper level user's management to the system?

1 Extremely enthusiastic.

3 Adequate.

5 Somewhat reluctant, or unknown.

7D.4 TECHNICAL ENVIRONMENT.

The following Complexity factors should affect mainly technical user oriented segments such as technical design, programming, etc.

T.1 Computer type, operating system, installation aids and project familiarity.

1 Computer with simple operating system, good installation aids and project team is familiar with it.

2 Computer with average operating system, fair installation aids and all are generallyfamiliar to project team.

5 Computer with average operating system, fair installation aids and project team is unfamiliar with.

5 Complex, large computer and operating system, includes installation aids which are generally familiar to the project team.

5 Complex, large computer and operating system which are not familiar to the project team.

T.1.2 Is any hardware new to the company?

0 None.

2 Most hardware is familiar to the project team.

4 Most hardware is familiar to a part of the project team.

5 Most hardware is new to the entire project team.

5 All hardware is new to the entire project team.

Only the highest score between T.1 and T.1.2 is used in calculating the complexity value for the 'technical environment indicator'.

T.2 Data-base management system.

1 Easy to use. Much prior experience.

3 Typical major DBMS with prior experience.

5 New or user customised.

T.3 On line monitor.

1 Easy to use. Much prior experience. Handles all system requirements.

3 Major monitor. No supplementary logic required.

5 New or user customised.

T.4 Data dictionary.

1 Data dictionary with development aids is available.

3 Data dictionary without development aids is available.

5 New dictionary or no dictionary.

T.5 The development methodology.

1 There is use of propriety structured methodology, for the whole life cycle.

3 There is no use of propriety structured methodology, but some usage of structured technique when designed.

5 There is only limited use of structured techniques.

5 There is no use of structure techniques.

T.6 Communication and distributed systems.

1 Established and there is no need for modification.

3 Established but some development required to support.

5 First time use.

If this question T.6 is scored for 5 (Communication and distributed systems first used) and one additional determinants which is included in this category is scored high (5),

than 'technical environment' should be considered as complex.

Or, if any three questions are scored 4 or 5,

than the system should be considered as complex.

T.7 **System architecture.**

This question was not found contributing in the sample projects used in this thesis.

1 Centralised (single processor).

2 Coupled (multiple processors).

3 Federated (processors linked via bus).

4 Distributed (centralised data-base).

5 Distributed (distributed data-base).

7D.5 PROJECT TEAM COMPOSITION

C.1 **Project team structure.**

1 Single decision maker and less than four team members.

2 Four to eight project team members. Some technical assistance is required.

5 Large project team and **multiple** decision makers.

5 Large team and **matrix organisation.**

5 Ambiguous or uncertain project organisation structure.

C.2 **Experience with industry/application.**

1 Extensive, high degree of capability.

3 Considerable, previous exposure but limited knowledge.

5 Minimal, first exposure.

C.3 **Technical experience**

1 Extensive, all required systems software has been successfully used before.

3 Considerable, small learning curve anticipated.

5 None, major learning curve.

C.4 **Staffing and Hiring.**

1 There was no need to hire new personnel for this project.

3 Personnel could be hired as justified, for this project.

5 There is a need to hire personnel for this project but hiring is difficult.

7D.6 ASSESSMENT OF PROJECT RISK

The following determinants are incorporated into the risk assessment:

R.1 Total effort in PM.

1 100 PM - 3000 PM.

2 3000 PM - 15000 PM.

3 15000 PM - 30000 PM.

5 30000 PM and more.

R.2 Project duration.

1 12 PM and less.

2 13 PM - 24 PM.

5 More then 24 PM.

Implementation time is calculated based on the coarse estimated, number of hours divided by full time people x 3, might be one way to produce coarse estimates of the project duration, another might be to incorporate the estimates of the EEM into a PERT tool.

S.1.2 System impact on financial status?

1 Minor.

3 Major.

5 Critical.

S.1.3 System impact on operational status?

1 Minor.

3 Major.

5 Critical.

S.2.1 Number of departments (other then the IS) involved with the project?

1 One.

3 Two.

5 Three or more.

S.2.3 If you propose to replace the system, what percentage of existing functions are replaced on a one-to-one basis?

1 50% to 100%.

3 25% to 50%.

5 0% to 25%.

S.2.4 What is the severity of procedural changes in the user department caused by the proposed system?

1 Low.

3 Medium.

5 High.

S.2.6 Does user organisation have to change structurally to meet requirements of the new system?

0 No.

1 Minimal.

3 Somewhat.

5 Major.

S.2.4 What is the severity of procedural changes in the user department caused by the proposed system?

1 Low.

3 Medium.

5 High.

S.2.6 Does user organisation have to change structurally to meet requirements of the new system?

0 No.

1 Minimal.

3 Somewhat.

5 Major.

S.2.7 What is the general attitude of user?

1 Good - understands value of data processing solution.

3 Fair - sometimes reluctant.

5 Poor - anti data processing solution.

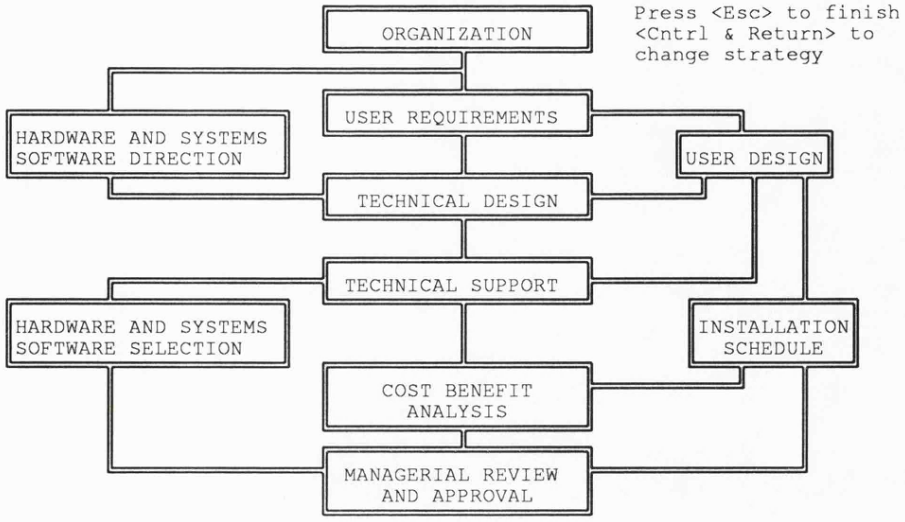
C.4 Staffing and Hiring.

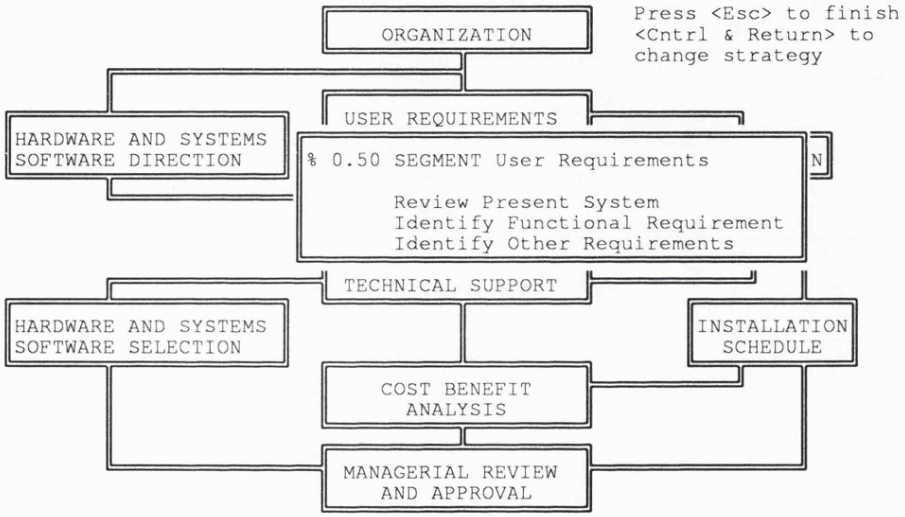
1 There was no need to hire new personnel for this project.

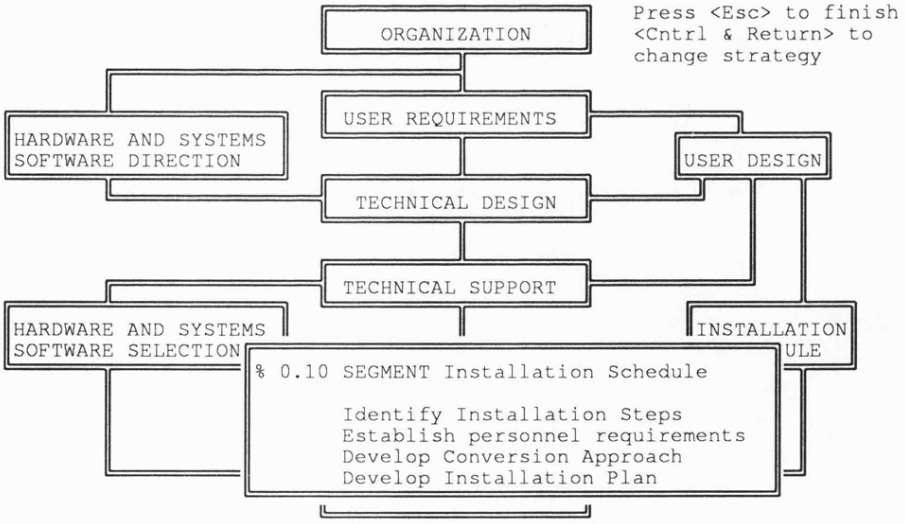
3 Personnel could be hired as justified, for this project.

5 There is a need to hire personnel for this project but hiring is difficult.

Appendix 7E. Case study 'A', example Screens


ORGANIZATION
Press <Esc> to finish
<Cntrl & Return> to
change strategy
USER REQUIREMENTS
HARDWARE AND SYSTEMS
SOFTWARE DIRECTION
USER DESIGN
TECHNICAL DESIGN
TECHNICAL SUPPORT
HARDWARE AND SYSTEMS
SOFTWARE SELECTION
INSTALLATION
SCHEDULE
COST BENEFIT
ANALYSIS
MANAGERIAL REVIEW
AND APPROVAL


ORGANIZATION
Press <Esc> to finish
<Cntrl & Return> to
change strategy
USER REQUIREMENTS
HARDWARE AND SYSTEMS
SOFTWARE DIRECTION
% 0.50 SEGMENT User Requirements
N
Review Present System
Identify Functional Requirement
Identify Other Requirements
TECHNICAL SUPPORT
HARDWARE AND SYSTEMS
SOFTWARE SELECTION
INSTALLATION
SCHEDULE
COST BENEFIT
ANALYSIS
MANAGERIAL REVIEW
AND APPROVAL


ORGANIZATION
Press <Esc> to finish
<Cntrl & Return> to
change strategy
USER REQUIREMENTS
HARDWARE AND SYSTEMS
SOFTWARE DIRECTION
USER DESIGN
TECHNICAL DESIGN
TECHNICAL SUPPORT
HARDWARE AND SYSTEMS
SOFTWARE SELECTION
INSTALLATION
ULE
% 0.10 SEGMENT Installation Schedule
Identify Installation Steps
Establish personnel requirements
Develop Conversion Approach
Develop Installation Plan

Prudent

Effort Results	Estimate	PERCENTAGE	Expected % Custom	Package
Organisation	32.00	2.3%	1-2	1-2
H and Sys Software sel	0	0 %	0-5	0-5
Application Software Eval	0	0 %	0	20-30
User Requirements	168.00	12.%	10-14	10-14
User Design	471.00	34.%	18-25	18-25
Technical Design	269.00	19.%	18-25	18-25
Tech and Func Spec	908.00	66.%		
Technical Support	288.00	20.%	5-7	5-7
Installation Schedule	64.00	4.6%	5	5
Cost Benefit Schedule	80.00	5.8%	5	5
Hardware and Software Selection	0	0 %	0-5	10-15
Total	1372.00	100%	100%	

Results without accounting for the life cycle changes.

Prudent
0.00 0.10

Alterations to be done	Estimate	*	+	Reason
Organisation	32.00	1.30	0	Users Involvement
H and Software selection	0	0	0	
Application Software Eval	0	1.00	0	
User Requirements	168.00	0.50	0	User Involvement
User Design	471.00	0.50	0	Users Involvment
Technical Design	269.00	0.25	0	Only 3.00 New Fil
Tech & Func Spec	908.00			
Technical Support	288.00	0.25	0	Conversion Only
Installation Schedule	64.00	0.10	0	H&s Already In Pl
Cost Benefit Schedule	80.00	0.80	0	part of C/B postp.
Hardware and Software Selection	0	1.00	0	
Total	1372.00			

The decisions taken through the life cycle editing are shown on this screen, but, the calculation of the effort has not taken them into account yet.

Alterations to be done	Estimate	*	+	Reason
Organisation	32.00	1.30	0	Users Involvement
H and Software selection	0	0	0	
A				
U What Do You Add To This Value				
U -20				
T				
T				
Installation Schedule	64.00	0.10	0	H&s Already In Pl
Cost Benefit Schedule	80.00	0.80	0	Part of C/B postp.
Hardware and Software Selection	0	1.00	0	
Total	1372.00			

Additional changes were requested, the screen formating was postponed to the next phase. The reason for this change is recorded on the next screen.

```
                Alterations to be done
                          Estimate   *    +    Reason
Organisation              |32.00    1.30 0    Users Involvement
H and Software selection  |0        0    0
A
U Why Do You Make This Adjustment
U Postponed scr. format.
T
T
Installation Schedule     |64.00    0.10 0    H&s Already In Pl
Cost Benefit Schedule     |80.00    0.80 0    Part of C/B postp.
Hardware and Software     |0        1.00 0
Selection                 |
==========================|====================================
Total                     |1372.00
```

```
                Alterations to be done
                          Estimate   *    +    Reason
Organisation              |32.00    1.30 0    Users Involvement
H and Software selection  |0        0    0
Application Software Eval |0        1.00 0
User Requirements         |168.00   0.50 0    User Involvement
User Design               |471.00   0.50 -20. Postponed Formatt
Technical Design          |269.00   0.25 0    Only 3.00 New Fil
  Tech & Func Spec        |908.00
Technical Support         |288.00   0.25 -20. Postp. Formmating
Installation Schedule     |64.00    0.10 0    H&s Already In Pl
Cost Benefit Schedule     |80.00    0.80 0    Part of C/B postp.
Hardware and Software     |0        1.00 0
Selection                 |
==========================|====================================
Total                     |1372.00
```

All the changes requested are shown on this screen.

```
                Effort Results                      Expected %
                          Estimate   PERCENTAGE  Custom Package
Organisation              |41.60        7.2%     1-2     1-2
H and Sys Software sel    |0            0  %     0-5     0-5
Application Software Eval |0            0  %     0       20-30
User Requirements         |84.00       14.%     10-14   10-14
User Design               |235.50      41.%     18-25   18-25
Technical Design          |67.25       11.%     18-25   18-25
  Tech and Func Spec      |386.75      67.%
Technical Support         |72.00       12.%     5-7     5-7
Installation Schedule     |6.40         1.1%    5       5
Cost Benefit Schedule     |64.00       11.%     5       5
Hardware and Software     |0            0  %     0-5     10-15
Selection                 |
==========================|====================================
Total                     |570.75      100%      100%
```

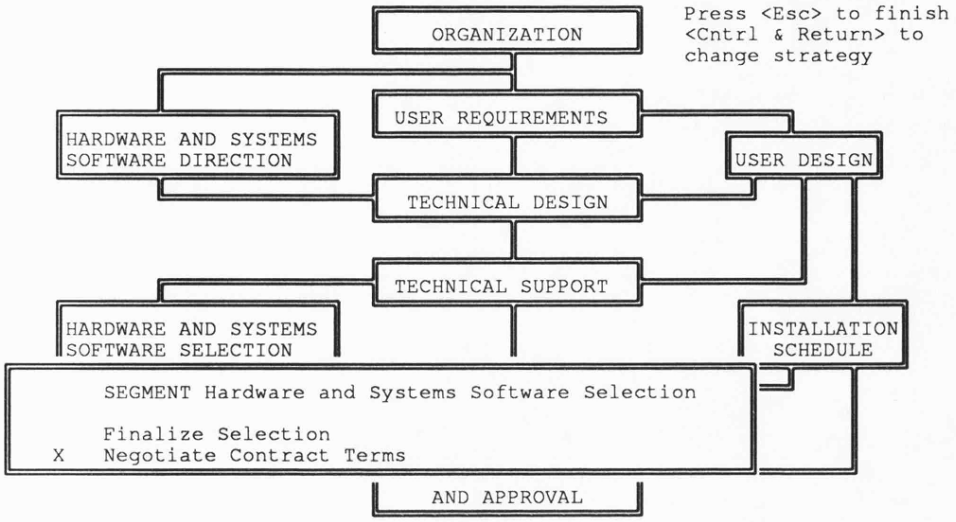
These are the results, considering the first set of assumptions taken at the life cycle editing session only.

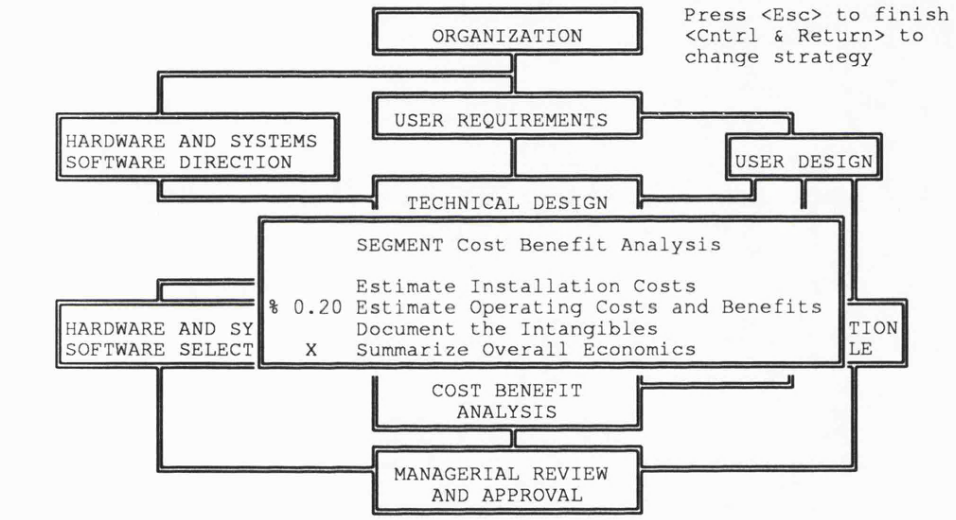
```
                Effort Results                      Expected %
                          Estimate   PERCENTAGE  Custom Package
Organisation              |41.60        7.8%     1-2     1-2
H and Sys Software sel    |0            0  %     0-5     0-5
Application Software Eval |0            0  %     0       20-30
User Requirements         |84.00       15.%     10-14   10-14
User Design               |215.50      40.%     18-25   18-25
Technical Design          |67.25       12.%     18-25   18-25
  Tech and Func Spec      |366.75      69.%
Technical Support         |52.00        9.8%    5-7     5-7
Installation Schedule     |6.40         1.2%    5       5
Cost Benefit Schedule     |64.00       12.%     5       5
Hardware and Software     |0            0  %     0-5     10-15
Selection                 |
==========================|====================================
Total                     |530.75      100%      100%
```

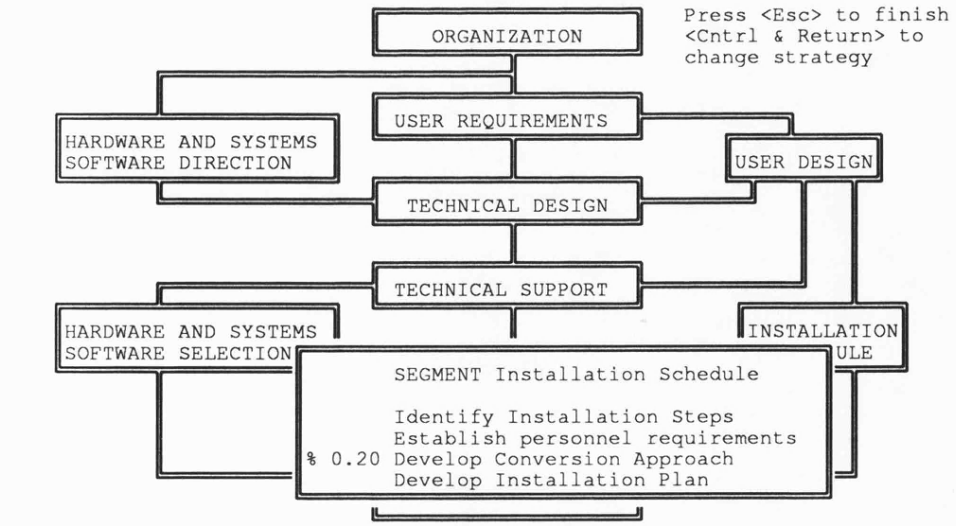
This is the final result. The effort estimated for the PSD in PH, considering all the assumptions made.

Appendix 7F. Case study 'B', example Screens

The life cycle assumptions are recorded by the EEM as shown on the following screens.







The effort required for the PSD by all parties associated with the development are shown below. This did not take into account the life cycle changes.

Effort Results	Estimate	PERCENTAGE	Expected % Custom	Package
Organisation	100.00	0.6%	1-2	1-2
H and Sys Software sel	252.00	1.6%	0-5	0-5
Application Software Eval	4080.00	26.%	0	20-30
User Requirements	1061.00	6.8%	10-14	10-14
User Design	1483.00	9.6%	18-25	18-25
Technical Design	1690.00	10.%	18-25	18-25
Tech and Func Spec	4234.00	27.%		
Technical Support	3064.00	19.%	5-7	5-7
Installation Schedule	102.00	0.6%	5	5
Cost Benefit Schedule	32.00	0.2%	5	5
Hardware and Software Selection	3580.00	23.%	0-5	10-15
Total	15444.0	100%	100%	

The effort required for the PSD, considering the changes to life cycle are shown below.

Effort Results	Estimate	PERCENTAGE	Expected % Custom	Package
Organisation	100.00	0.8%	1-2	1-2
H and Sys Software sel	262.00	2.1%	0-5	0-5
Application Software Eval	4080.00	32.%	0	20-30
User Requirements	1061.00	8.5%	10-14	10-14
User Design	1483.00	11.%	18-25	18-25
Technical Design	1695.00	13.%	18-25	18-25
Tech and Func Spec	4239.00	34.%		
Technical Support	0	0 %	5-7	5-7
Installation Schedule	102.00	0.8%	5	5
Cost Benefit Schedule	32.00	0.2%	5	5
Hardware and Software Selection	3580.00	28.%	0-5	10-15
Total	12395.0	100%	100%	

Additional changes were required. The changes required and the reasons for them are recorded on the following three screens.

Alterations to be done	Estimate	*	+	Reason
Organisation	100.00	1.00	0	
H and Software selection	252.00	1.00	0	
A				
U What Do You Multiply The Value By				
U 1.1				
T				
T				
Installation Schedule	102.00	1.00	0	
Cost Benefit Schedule	32.00	1.00	0	
Hardware and Software Selection	3580.00	1.00	0	
Total	15444.0			

Alterations to be done

	Estimate	*	+	Reason
Organisation	100.00	1.00	0	
H and Software selection	252.00	1.00	0	
A				
U	Why Do You Make This Adjustment			
U	anticipated difficulty			
T				
T				
Installation Schedule	102.00	1.00	0	
Cost Benefit Schedule	32.00	1.00	0	
Hardware and Software Selection	3580.00	1.00	0	
Total	15444.0			

Alterations to be done

	Estimate	*	+	Reason
Organisation	100.00	1.00	0	
H and Software selection	252.00	1.00	0	
Application Software Eval	4080.00	1.00	0	
User Requirements	1061.00	1.10	0	Anticipated Diffi
User Design	1483.00	0.85	0	Use Of Report Gen
Technical Design	1690.00	1.00	0	
Tech & Func Spec	4234.00			
Technical Support	3064.00	0	0	
Installation Schedule	102.00	1.00	0	
Cost Benefit Schedule	32.00	1.00	0	
Hardware and Software Selection	3580.00	1.00	0	
Total	15444.0			

The final results are shown below.

Effort Results

	Estimate	PERCENTAGE	Expected % Custom	Package
Organisation	100.00	0.8%	1-2	1-2
H and Sys Software sel	252.00	2.0%	0-5	0-5
Application Software Eval	4080.00	33.%	0	20-30
User Requirements	1167.10	9.5%	10-14	10-14
User Design	1260.55	10.%	18-25	18-25
Technical Design	1690.00	13.%	18-25	18-25
Tech and Func Spec	4117.65	33.%		
Technical Support	0	0 %	5-7	5-7
Installation Schedule	102.00	0.8%	5	5
Cost Benefit Schedule	32.00	0.2%	5	5
Hardware and Software Selection	3580.00	29.%	0-5	10-15
Total	12263.6	100%	100%	

The cost drivers associated with project 'B' are shown on the following two screens.

Cost Drivers (1 of 2)	
Number of new systems	1.00
Number of project team members	5.00
Number of interviews	50.0
Number of old reports,screens,inputs	170.
Number of new reports	50.0
Number of new screens,I/O,messages	120.
Number of new inquiries	2.00
Number of new or redesigned forms	4.00
Number of data elements (/100)	7.20
Number of complex functions	6.00
Number of complex processes	2.00

Cost Drivers (2 of 2)	
Number of utility/control modules	12.0
Number of file conversion processes	0
Number of files to be converted	4.00
Number of application packages	0
Number of modifications to packages	0
Number of invitations to tender	5.00
Number of requests for proposals	17.0
Number of contracts to be negotiated	0
Number of high volume inputs	110.
Number of simulation models	0
Number of iterations for prototyping	0

BIBLIOGRAPHY

[Abd89]
Abdel-Hamid, T.K., and Madnick, S.E. "Lessons Learned From Modelling the Dynamics of Software Development". *Communications of the ACM*, Vol. 32, No. 12, (December 1989), pp. 1426-1455.

[Aco87]
Acosta, E.C., and Golub, H.D. "Early Experience in Estimation of Embedded Software Cost at AVSCOM". *Proceedings of the International Society of Parametric Analysis, 9th Annual Conference.* Vol. VI, No. 1, San Diego, CA., (May 1987), pp. 665-675.

[Agr86]
Agresti, W.W. "The Conventional Software Life-Cycle Model: Its Evolution and Assumptions". Printed in: *Tutorial on New Paradigms for Software Development*, edited by: Agresti, W.W., pp. 2-5. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Agr86a]
Agresti, W.W. "Framework for a Flexible Development Process". Printed in: *Tutorial on New Paradigms for Software Development*, edited by: Agresti, W.W., pp. 11-14. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Ala84]
Alavi, M. "An Assessment of the Prototyping Approach to Information System Development". *Communications of the ACM*, (June 1984), pp. 556-563.

[Alb79]
Albrecht, A.J. "Measuring Application Development Productivity". *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium.* Chicago,

Guide International Corporation, 1979.

[Alb83]
Albrecht, A.J., and Gaffney, J.E. "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation". *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, (1983), pp. 639-648.

[Alb84]
Albrecht, A.J. "AD/M Productivity Measurement and Estimate Validation-Draft". *IBM Information Systems and Administration, AD/M Improvement Program.* Purche, NY., (May 1984).

[And79]
Arthur Andersen. *Method/1: Systems Development Practice.* Chicago, IL.: Arthur Andersen & Co., 1979.

[Aro69]
Aron, J.D. "Estimating Resources for Large Systems". In: *Software Cost Estimating and Life Cycle Control,* edited by: Putnam, L.H., pp. 257-268. IEEE Computer Society, 1980.

[Aro74]
Aron, J.D. *The Program Development Process: Part I - The Individual Programmer.* Reading, MA.: Addison- Wesley, 1974.

[Bai81]
Bailey, J.W., and Basili V.R. "A Meta-Model for Software Development and Resource Expenditures". *Proceedings of the 5th International Conference on Software Engineering.* New York: IEEE, (March 1981), pp. 107-116.

[Bal81]
Balzer, R. "Transformational Implementation: An Example". *IEEE Transactions on Software Engineering,* (January 1981), pp. 3-14. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti, W.W., pp. 227-238. IEEE

Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Bal82]
Balzer, R.M., Goldman, N.M., and Wile, D.S. "Operational Specification as the Basis for Rapid Prototyping". *ACM SIGSOFT Software Engineering Notes*, Vol. 7, No. 5, (December 1982), pp. 3-16. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti, W.W., pp. 116-132. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Bal83]
Balzer, R., Cheatham, T.E., and Green, C. "Software Technology in the 1990's: Using a New Paradigm". *IEEE Computer*, Vol. 16, No. 3, (March 1983), pp. 39-45.

[Ban88]
Banker, R.D., and Kemerer, C.F. "Scale Economies in New Software Development". *Center for Information System Research, Sloan School Of Management MIT.* CISR WP No. 167, (February 1988).

[Bas75]
Basili, V.R., and Turner, A.J. "Iterative Enhancement: A Practical Technique of Software Development". *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, (December 1975), pp. 390-396.

[Bas78]
Basili, V.R., and Zelkowitz, M.V. "Analysing Medium Scale Software Developments". *Proceedings of the 3th International Conference on Software Engineering,* (May 1978).

[Bas79]
Basili, V.R., and Zelkowitz, M.V. "Measuring Software Development Characteristics in the Local Environment". *Computer and Structures*, 10, (1979).

[Bas79a]
Basili, V.R. "Quantitative Software Complexity Models: A Panel Summary".

Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art. IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 243-245.

[Bas79b]
Basili, V.R., and Reiter, R.W. Jr. "An Investigation of Human Factors in Software Development". *IEEE Computer*, Vol. 12, No. 12, (December 1979), pp. 21-38.

[Bas80]
Basili, V.R. "Resource Models". *Tutorial on Models and Metrics for Software Management and Engineering*, edited by: Basili V.R. IEEE (1980), pp. 4-9.

[Bas81]
Basili, V.R., and Beane, J. "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?" *Journal of Systems and Software,* Vol. 2, No. 1, (February 1981), pp. 59-69.

[Bas81a]
Basili, V.R., and Freburger, K. "Programming Measurement and Estimation in the Software Engineering Laboratory". *Journal of Systems and Software,* Vol. 2, (1981), pp.47-57.

[Bas81b]
Basili, V.R., and Philips, T. "Evaluating and Comparing the Software Metrics in the Software Engineering Laboratory". *Performance Evaluation Review. ACM SIGMETRICS,* 10, (March 1981), pp. 95-106.

[Bas81c]
Basili, V.R., and Reiter, R.W. Jr. "A Controlled Experiment: Quantitatively Comparing Software Development Approaches". *IEEE Transactions on Software Engineering,* Vol. SE-7, No. 3, (1981), pp. 229-320.

[Bas83]
Basili, V.R. "Resource Models". Printed in: *Software Metrics,* edited by: Perlis, A.,

Sayward F.G., and Shaw, M., pp. 111-130. Cambridge MA.: The MIT Press, 1983.

[Bau72]
Bauer, F.L. *Software Engineering.* Amsterdam: North Holland, 1972.

[Bau82]
Bauer, F.L. "From Specifications to Machine Code: Program Construction through Formal Reasoning". *Proceedings of the 6th International Conference on Software Engineering,* (1982), pp. 84-91.

[Beh83]
Behrens, C.A. "Measuring the Productivity of Computer Systems Development Activities with Function Points". *IEEE Transactions on Software Engineering,* Vol. SE-9, No. 6, (November 1983), pp. 648-652.

[Bel71]
Belady, L.A., and Lehman, M.M. "Programming Systems Dynamics, or the Metadynamics of Systems in Maintenance and Growth". *IBM Research Report* RC3546, (September 1971).

[Bel76]
Belady, L.A., and Lehman, M.M. "A Model of Large Program Development". *IBM Systems Journal,* Vol. 15, No.3, (1976), pp. 225-252.

[Bel77]
Belady, L.A., and Merlin, P.M. "Evolving Parts and Relations - A Model of System Families", *IBM Research Report* RC6677, (August 1977).

[Bel79]
Belady, L.A. "An Anti Complexity Experiment". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art,* edited by: Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 128-129.

[Bel79a]
Belady, L.A. "On Software Complexity". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art,* edited by: Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 90-94.

[Bel79b]
Belady, L.A., and Lehman, M.M. "The Characteristics of Large Systems". Printed in: *Research Directions in Software Technology*, edited by: Wegner, P., pp. 106-139. Cambridge, MA.: The MIT Press, 1979.

[Bel81]
Belady, L.A. "Complexity of Large Systems". Printed in: Software Metrics: An Analysis and Evaluation. Edited by: Perlis, A., Sayward F.G., and Shaw M., pp. 225-234. Cambridge, MA.: The MIT Press, 1981.

[Ben56]
Benington, H.D. "Production of Large Computer Programs". *Proceedings of Symposium on Advanced Computering for Digital Computers*, (1956). Republished in: *Annals of the History of Computing,* (October 1983), pp. 350-361.

[Bla77]
Black, R.K.D., Curnow, R.P., Kats, R., and Gray, M.D. "Final Technical Report, RADC-TR-77-116, Boeing Computer Services". *BCS Software Production Data,* NTIS, No. AD-A039852, (March 1977).

[Blu69]
Blumental, S.C. *Management Information Systems; A Framework for Planning and Development.* New Jersey, N.J.: Prentice Hall, Englewood Cliffs, 1969.

[Blu84]
Blum, I.B. "Three Paradigms for Developing Information Systems". *Proceedings of the 7th International Conference on Software Engineering,* (1984), pp. 534-543.

[Boe73]
Boehm, B.W. "Software and its Impact: A Quantitative Assessment". *Datamation*, (May 1973), pp. 48-59.

[Boe76]
Boehm, B.W. "Software Engineering". *IEEE Transactions on Computers*, Vol. C-25, No. 12, (December 1976), pp. 1226-41. Reprinted in: *Classics in Software Engineering*, edited by: Yourdon, E., pp. 325-61. New York: Yourdon Press, 1979.

[Boe76a]
Boehm, B.W., Brown J.R., and Lipow M. "Quantitative Evaluation of Software Quality". *Proceedings of the 2th International Conference on Software Engineering*, (1976), pp. 592-605.

[Boe80]
Boehm, B.W. "Software Engineering: R&D Trends and Defence Needs". Printed in: *Research Directions in Software Technology*, edited by: Wegner, P., pp. 44-86. Cambridge, MA.: MIT Press, 1980.

[Boe81]
Boehm, B.W. *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[Boe82]
Boehm, B.W. "Keeping a Lid on Software Costs". *Computer World*, (28 January 1982).

[Boe84]
Boehm, B.W., Gray T., and Seewaldt T. "Prototyping Versus Specifying: A Multi project Experiment". *IEEE Transaction on Software Engineering*. Vol. 10, No. 3, (May 1984), pp. 290-303.

[Boe87]
Boehm, B.W., and Belz F.C. "Reasoning About Iteration: A Cost Benefit

Approach". *Proceedings of the 3th International Software Process Workshop.* IEEE Computer Society Press, (1987), pp. 40-42.

[Boe87a]
Boehm, B.W. "Improving Software Productivity". *IEEE Computer*, (September 1987), pp. 43-54

[Boe88]
Boehm, B.W. "A Spiral Model of Software Development and Enhancement". *IEEE Computer,* (May 1988), pp. 61-72.

[Boe88a]
Boehm, B.W. "Understanding and Controlling Software Costs". *IEEE Transactions on Software Engineering,* Vol. 14, No. 10, (October 1988), pp. 1462-1477.

[Boy84]
Boydston, R.E. "Programming Cost Estimate: Is It Reasonable?". *Proceedings of the 7th Conference on Software Engineering,* (1984), pp. 153-160.

[Boz84]
Bozoki, G.J. *SSM User Guide.* GJB Associates, 1984.

[Boz87]
Bozoki, G.J. *Target Software*, (May/June 1987)

[Bra63]
Brandon, D.H. *Management Standards for Data Processing.* New York: Van Nostrad, 1963.

[Bro75]
Brooks, F. *The Mythical Man Month.* New York: Addison-Wesley, 1975.

[Bro83]
Browne, J.C., and Shaw, M. "Toward a Scientific Basis for Software Evaluation". Reprinted in: *Software Metrics: An Analysis and Evaluation,* edited by: Perlis, A., Sayward F.G., and Shaw M., pp. 19-42. Cambridge, MA.: The MIT Press, 1983.

[Bro87]
Brooks, F. B. "No Silver Bullet, Essence and Accidents of Software Engineering". Printed in: *Information Processing '86,* edited by: Kugier, H. J. ISBN No. 0-444-70077-3. Elsevier Science Publishers B.V. North Holland. IFIP, (1986). Reprinted in: *Computer,* (April 1987), pp. 10-19.

[Bud84]
Budde, R. *Approaches to Prototyping,* edited by: Budde, R., Kuhlenkamp, K., Mathiassen, L., and Zullighoven, H. Springer-Verlag, Heidelberg, 1984.

[Cal84]
Callisen, H., and Colborne, S. "A proposed Method for Estimating Software Cost from Requirements". *Journal of Parametric,* Vol. IV, No. 4, (December 1984), pp. 33-40.

[Cha74]
Channon, R.N. "On a Measure of Program Structure". *Proceedings of the Programming Symposium*, edited by: Goos, G., and Hartmanis, J. Paris: Springer-Verlag, 1974.

[Che77]
Chen, P. P. "The entity relationship Approach to Logical Data-base Design". *The Q.E.D. Monograph Series, Data-base Management,* No. 6. Q.E.D. Information Science, Inc., 1977.

[Che78]
Chen, E.T. "Program Complexity and Programmer Productivity". *IEEE Transactions on Software Engineering,* Vol. SE-4, No. 3, (May 1978), pp. 187-94.

[Che81]
Cheatham, T.E. Jr., Holloway, G.H., and Townley J.A. "Program Refinement by Transformation". *Proceedings of the 5th International Conference on Software Engineering*, (1981), pp. 430-437.

[Chr78]
Chrysler, E. "Some Basic Determinants of Computer Programming Productivity. *Communications of the ACM*, Vol. 21 No. 6, (June 1978) pp. 472-483.

[Coh88]
Cohen, G., Mancini, L., Mathieu, A., and Mora, A. "Data Classification and Measurement Procedure". SPEM Esprit project. Deliverable D1.3, (March 1988).

[Con76]
US Congress, House of representatives, "Advanced Logistic (ADP) Systems," *Department of Defence, Appropriation Bill, 1976,* Report No. 94-517, (September 25, 1975), pp. 163-165.

[Con86]
Conte, S.D., Dunsmore, D.E., and Shen, V.Y. *Software Engineering Metrics and Models*. Menlo Park, CA.: Benjamin/Cummings, 1986.

[Cor80]
Cordata, J.W. *EDP Cost and Charges.* Prentice-Hall Publication, 1980.

[Cow86]
Cowderoy, A.J.C., and Jenkins, J.O. "State of the Art Survey for Software Cost-estimation". *ESPRIT P38* report WP5 1a (issue 3). London: Imperial College Management School, (1986).

[Cow87]
Cowderoy, A.J.C., and Jenkins, J.O. "New Trends in Cost Estimation". *The 4th Annual Conference of the Centre of Software Reliability*. Elsevier, 1987

[Cow88]
Cowderoy, A.J.C., and Jenkins, J.O. "Cost-estimation by Analogy as Good Management Practice". *Proceedings of the 2nd IEE/BCS conference on Software Engineering.* Liverpool, (1988).

[Cow88a]
Cowderoy, A.J.C., and J.O. Jenkins. "Combining Expert System Technology with Parameters for Cost Estimation". *Proceedings of the International Society of Parametric Analysis,* Vol. VII, (1) (July 1988), pp. 149-163.

[Cow88b]
Cowderoy, A.J.C., and Harry, C.M., and Jenkins, J.O. "Software Process Modelling", London: Imperial College, Management School ,(1988).

[Cow89]
Cowderoy, C., Jenkins, J., and Levy, Z. "State of The Art Survey, Effort and Size Estimation Models". MERMAID Esprit project P2046. Deliverable D1.2C, Vol. 1 (February 1989).

[Cue87]
Cuelenere, A.M.E., and Genuchten, M.J., and Heemstra, F.J. "Calibrating Cost Estimation Model: Why and How?". *Information and Software Technology*, Vol. 29, No. 10, (December 1987).

[Cur79]
Curtis, B. "In Search of Software Complexity". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art,* edited by: Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 95-106.

[Cur81]
Curtis, B. "The Measurement of Software Quality and Complexity". Printed in: *Software Metrics: An Analysis and Evaluation,* edited by: Perlis, A., Sayward F.G., and Shaw, M., pp. 203-224. Cambridge, MA.: The MIT Press, 1981.

[Cur81a]
Curtis, B. "Experimental Evaluation of Software Characteristics". Printed in: *Software Metrics: An Analysis and Evaluation,* edited by: Perlis, A., Sayward F.G., and Shaw, M., pp. 61-76. Cambridge MA.: The MIT Press, 1981.

[Cur87]
Curtis, B., and Knasner, S.V., and Jscoe, N. "On Building Software Process Models Under the Lamppost". *Proceedings of the 9th International Conference on Software Engineering,* (1987), pp. 96-103.

[Dal85]
Dale, C.J., and Kitchenham, B.A. *Report on visit to USA Software data library report,* 1985.

[DeM77]
Demarco, T. *Report on the 1977 Productivity Survey.* New York: Yourdon Inc., (September 1977).

[DeM82]
DeMarco, T. *Controlling Software Projects: Management Measurement and Estimation.* New York: Yourdon, 1982.

[DeM85]
DeMarco, T., and Lister T. "Programmer Performance and the Effects of the Workplace". *Proceedings of the 8th International Conference on Software Engineering.* IEEE, (August 1985), pp. 268-272.

[Des88]
Desharnais, J. M. "Analyse statistique de la productive des projets de development en informatique a partir de la technique des points de function". Rapport d'activite de synthese, Programme de maitrise en informatique de gestion, *Universite du Quebec a Montreal,* (December 1988).

[Deu79]
Deutsch, M. S. "Verification and Validation". Printed in: *Software Engineering,* edited by: Jensen, R.W., and Tonies C.C., pp. 329-407. Prentice-Hall Publications, 1979.

[Dij75]
Dijkstra, E. W. "Guarded commands, nondeterminancy, and formal derivation of programs", *Communications of the ACM,* (8), (August 1975), pp. 453-457.

[Dix88]
Dixon, D. "Integrated Support for Project Management". *Proceedings of the 10th International Conference on Software Engineering,* (1988), pp. 49-58.

[DOD85]
"DOD Computing Activities and Programs: Ten Year Market Forecast Issues, 1985-1995". *Electronic Industries Association,* (October 1985).

[Dot77]
Doty, D.L., Nelson, P.J., and Stewart, K.R. "Software Cost Estimation Study: Guidelines for Improved Software Cost Estimating". *Technical Report RADC-TR-77-220,* Vol. II. Rome Air Development Center, (August 1977).

[Dow87]
Dowson, M., "Iteration in the Software Process", *Proceedings of the 9th International Conference on Software Engineering.* IEEE, (1987), pp. 36-39.

[Dun83]
Dunham, J.R., and Kruesi, E. "The Measurement Task Area". *IEEE Computer,* Vol. 16, No. 11, (November 1983), pp. 47-54.

[Els78]
Elshoff, J.L. "An Investigation into the Effect of the Counting Method Used on Software Science Measurements". *Proceedings of the 5th International Conference on Software Engineering,* (1979), pp. 30-45.

[Fai85]
Fairley, R. *Software Engineering Concepts.* McGraw-Hill, New York, 1985.

[Far65]
Farr, L., and Zagorsky, H.J. "A Quantitative Analysis of Programming Cost Factors: A Progress Report". Printed in: *Economics of Automatic Data Processing - ICC Symposium Proceedings,* Rome 1965. Amsterdam: North-Holland, 1965.

[Fit78]
Fitzsimmons, A., and Love, T. "A Review and Evaluation of Software Science". *Computing Surveys*, Vol. 10, No. 1, (March 1978), pp. 3-18.

[Flo84]
Floyed, C. "A Systematic look at Prototyping". Printed in: *Approaches to prototyping*, edited by:, Budde, R., Kuhlenkamp, K., Mathiassen, L., and Zullighoven, H. Springer-Verlag, Heidelberg, 1984.

[Fox82]
Fox, J.M. *Software and Its Development.* New Jersey, N.J.: Englewood Cliffs, Prentice Hall Inc., 1982.

[Fre75]
Freeman, P. "Towards Improved Review of Software Designs". *Proceedings of the National Computer Conference.* AFIPS Press, (1975). Reprinted in: *Tutorial on Software Design Techniques*, edited by: Freeman, P., and Wasserman, A.I. IEEE Computer Society, IEEE Catalog No. EHO 161-0, (1980), pp. 434-439.

[Fre76]
Freeman, P., Wasserman, A.I and Fairley, R.E. "Essential Elements of Software Engineering Education". *Proceedings of the 2nd International Conference on Software Engineering.* October, (1976), pp. 116-122.

[Fre79]
Freburger, K., and Basili, V.R. "The Software Engineering Laboratory:

Relationship equations". University of Maryland, *Technical Report TR-764*, (May 1979).

[Frei79]
Freiman, F.R., and Park, R.E. "Price Software Model - Version 3. An Overview". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art*, edited by Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 32-41.

[Fun87]
Funch, P. "Recalibration of Intermediate COCOMO to recent AF Acquisitions". *Third Annual COCOMO Users' Group Meeting, Carnegie Mellon University,* Software Engineering Institute, Pittsburgh, PA, (November 3-5, 1987).

[Gaf79]
Gaffney, J.E. Jr. "Program Control Complexity and Productivity". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art*, edited by: Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979).

[Gaf79a]
Gaffney, J.E. Jr., and Heller, G.L. "Macro Variable Software Models for Application to Improved Software Development Management". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art*, edited by: Shooman, M.L. IEEE Catalog No. TH0067-9. New York: IEEE, (1979).

[Gaf84]
Gaffney, J.E. Jr. "Estimation of Software Code Size Based on Quantitative Aspects of Function (With Application of Expert Systems Technology)". *Journal of Parametrics,* Vol. 4, No. 3, (September 1984).

[Gal86]
Galashan, M.M. "Expert Systems in Software Cost-Estimation". MSc dissertation.

Kingston Polytechnic. London, (1986).

[Gal88]
Galorath, D.D., and Rampton, J.C. "Software Engineering Costing: Transition into the 90's". *ISPA, International Conference on Parametric Analysis*, Vol. VII, No. 1, (July 1988), pp. 253-258, .

[Geh82]
Gehani, N.H. "A Study in Prototyping", *ACM SIGSOFT Software Engineering Notes*, Vol. 7, No. 5, (December, 1982), pp. 71-74.

[Gil77]
Gilb, T. *Software Metrics.* Cambridge, MA.: Winthrop Publishers, 1977.

[Gil85]
Gilb, T.J. "Estimating Software Attributes: Some Unconventional Points of View. Unpublished article. Contact author: Iver Holtersvei 2, N-1410 Kolbotn, Norway, (1985).

[Gil87]
Gilb, T.J. The Principles of Software Engineering Management. Addison Wesley, 1987.

[Gom81]
Gomaa, H., and Scott, D.B.H. "Prototyping as a Tool in the Specification of User Requirements". *Proceedings of the 5th International Conference on Software Engineering.* IEEE, (1981), pp. 333-342. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti, W.W., pp. 69-77. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Gom83]
Gomaa, H. "The Impact of Rapid Prototyping on Specifying User Requirements", *ACM SIGSOFT Software Engineering Notes,* Vol. 8, No. 2, (April 1983), pp. 17-28.

[Gor77]
Gordon, R.L., and Lamb, J.C. "A Close Look at Brook's Law". *Datamation,* Vol. 23, No. 6, (June 1977), pp. 81-83.

[Gor87]
Gordon Group, *Before You Leap, User's Guide,* (1987).

[Hal72]
Halstead, M.H. "Natural laws controlling algorithm structure". *SIGPLAN Notices,* 7 (2), (1972), pp. 19-26.

[Hal77]
Halstead, M.H. *Elements of Software Science.* New York: Elsevier North-Holland, 1977.

[Her77]
Herd, J.H., Postak, J.N., Russekk, W.E., and Stewart, R.K. "Software Estimating Study; Study Results". *Technical Report RADC-TR-77-220* Rome Air Development Center, Vol. I, (June 1977).

[Hum39]
Hume, D. *The Treatise of Human Nature*, edited by: Selby-Biggs, L.A. Oxford: The Clarendon Press, 1978. Originally published, 1739.

[Hum88]
Humphreys, P. "Intelligence in Decision Support - A Process Model". Unpublished Paper. London School of Economics and Political Science, (1988).

[Hum89]
Humphreys, P. "Risk analysis tools and techniques in project management". *Proceedings of the Royal Society Symposium on Project Risk in space industry*, (March 1989).

[IIT87]
IIT Research Institute. *Software Cost Model Research.* US Army Cost and Economic Analysis Center (September, 1987).

[IIT87a]
IIT Research Institute. *A Descriptive Evaluation of Software Sizing Models.* IIT Research Institute; Data and Analysis Centre for Software. Lanham, Maryland, (1987).

[Ita81]
Itakura, M., and Takayanagi, A. "A Model for Estimating Program Size and Its Evaluation". *Proceedings of the 6th International Conference on Software Engineering,* (1981), pp. 104-109.

[Jef79]
Jeffery, D.R., and Lawrence, M.J. "An Inter-Organisational Comparison of Programming Productivity". *Proceedings of the 4th International Conference on Software Engineering,* (1979), pp. 369-377.

[Jef80]
Jeffery, D.R., and Vessey, I. "Models, Metrics and Management of IS Development". *Information and Management.* 3, (1980), pp. 89-93.

[Jef81]
Jeffery, D.R., and Lawrence, M.J. "Some Issues in the Measurement and Control of Programming Productivity". *Information and Management.* 4, (1981), pp. 169-176.

[Jef85]
Jeffery, J.R., and Lawrence, M.J. "Managing Programming Productivity". *The Journal of Systems and Software.* 5, 1, (February 1985), pp. 49-58.

[Jef88]
Jeffery, J.R., and Basili, V.R., "Validating the Tame Resource Data Model",

Proceedings of the 10th International Conference on Software Engineering, (1988), pp. 187-197.

[Jef88a]
Jeffery, D.R., Loo, C.C., and Low, G.C. "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).

[Jef89]
Jeffery, D.R., and Low, G.C. "Generic Estimation tools in the management of software development". Pre-publication draft. *School of Information Systems Research Report,* Department of Information Systems, University of New South Wales, (1989).

[Jen79]
Jensen, R.W., and Tonies, C.C. "Software Engineering Education: A Constructive Criticism". Printed in: *Software Engineering,* edited by: Jensen, R.W., and Tonies C.C., pp. 553-567. Prentice-Hall Publications, 1979.

[Jen79a]
Jensen, R.W., and Tonies, C.C. *Software Engineering,* edited by: Jensen, R.W., and Tonies C.C. Prentice-Hall Publications, 1979.

[Jen83]
Jensen, R.W. "An improved Macrolevel Software Development Resource Estimation Model". Hughes Aircraft Company, Space and Communications Group, Los Angeles, CA., (1983).

[Jen83a]
Jensen, R.W., and Lucas, S. "Sensitivity Analysis of the Jensen Software Model", Hughes Aircraft Company, Space and Communications Group, Los Angeles, CA., (1983).

[Jen84]
Jensen, R.W. "A Comparison of the Jensen and COCOMO Schedule and Cost Estimation Models". *Proceedings of the International Society of Parametric Analysis.* (1984), pp. 96-106.

[Jen86]
Jensen, R.W. "Predicting and Controlling Software Development Costs: Metrics for Managers", (October 1986)

[Joh77]
Johnson, J.R. "A Working Measure of Productivity". *Datamation,* (February 1977).

[Jon83]
Jones, T.C. "Demographic and technical trends in the Computing Industry". *Software Productivity Research, INC.,* (July 1983).

[Jon86]
Jones, T.C. *Programming Productivity.* McGraw-Hill, 1986.

[Kel1889]
Lord Kelvin, *Popular Lectures and Addresses,* 1889. In: Cook, M. L. Software Metrics: An Introduction and Annotated Bibliography. *Software Engineering Notes ACM SIGSOFT,* Vol. 7 No. 2, (April 1982), pp. 41-60.

[Kem87]
Kemerer, C.H. "An Empirical Validation of Software Cost Estimation Models". *Communications of the ACM,* Vol. 30, No. 5, (May 1987), pp. 416-428.

[Keu82]
Keus, H.E. "Prototyping : A More Reasonable Approach to System Development" *ACM SIGSOFT Software Engineering Notes,* Vol. 7, No. 5, (December 1982), pp. 94-95.

[Kit84]
Kitchenham, B.A., and Taylor, N.R. "Software Cost Models", ICL Technical Journal, (May 1984), pp. 73-102.

[Kit85]
Kitchenham, B.A., and Taylor, N.R. "Software Project Development Cost Models". *ICL Technical Journal.* (May 1984), pp. 73-102. Reprinted in: *The Journal of Systems and Software*, Vol. 5, No. 4, (May 1985), pp. 502-506.

[Kit85a]
Kitchenham, B.A. "Management Metrics". *CSR Workshop*, (1985).

[Kit87]
Kitchenham, B.A. "Management Metrics". Printed in: *Software reliability: Achievement and assessment.* Littlewood B., Blackwell Scientific Publication, Oxford, 1987.

[Kit89]
Kitchenham, B.A., and De Newmann, B. *Software reliability handbook.* Unpublished draft. Elsevier, 1989.

[Knu74]
Knuth, D.E. *Structure Programming with GOTO Statements. ACM* Computing Survey Vol. 6, No. 4, (December 1974), pp. 261-301.

[LaF87]
LaFourcade, D., and Pickford, B. "Counting Function Points for Continued Development. *International Function Point Users Group conference Proceedings*, (September 1987).

[Lam86]
Lambert, J.M. "A Software Sizing Model". *Journal of Parametrics*, Vol. VI. No. 4, (December 1986).

[Las79]
Lasher, W. "Software Cost Evaluation and Estimation: A Government Source Selection Case Study". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art.* IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 42-55.

[Law85]
Law, D. *Prototyping: A State of the Art Report*, NCC, 1985.

[Lec67]
Lecht, C.P. *The Management of Computer Programming Projects.* American Management Association, New York, 1967.

[Leh84]
Lehman, M.M., Stenning, V., and Turski, W.M. "Another Look at Software design Methodology". *ACM SIGSOFT Software Engineering Notes,* Vol. 9, No. 2, (April 1984), pp. 38-53.

[Leh85]
Lehman, M.M., and Belady, L.A. *Program Evolution.* London: Academic Press, 1985.

[Leh85a]
Lehman, M.M. "Approach to a Disciplined Development Process - The ISTAR Integrated Project Support Environment". ICST DoC Research Report 85/19. *Presentation to the Second Process Workshop,* Coto de Caza, (March 27-29, 1985).

[Leh85b]
Lehman, M.M., and Stenning, N. V. *Concepts of an Integrated Project Support Environment.* Butterworth & Co Ltd., 1985.

[Leh87]
Lehman, M.M. "Process Models, Process Programs, Programming Support". The *Proceedings of the 9th International Conference on Software Engineering,* (1987), pp. 14-16.

[Leh87a]
Lehman, M.M. "Model based approaches to ISPE architecture and the design. The 1ST ISTAR Project as an Instantiation". Invited Contribution. *Quarterly Bulletin IEEE,* (1987).

[Leh89]
Lehman, M.M. "Uncertainty in Computer Application and its Control Through the Engineering of Software". *Software maintenance: research and Practice,* John Wiley & Sons Ltd., Vol. 1, 3-27, 1989.

[Leh89a]
Lehman, M.M. "Software, Systems and Application Uncertainty and its Control Through the Engineering of Software". Lehman Software Technology Associates Ltd., and Department of Computing Imperial College of Science and Technology. Unpublished paper, (November 1989).

[Mac87]
Macro, A., and Buxton, J. *The Craft of Software Engineering.* UK: Addison-Wesley Publishing Co., 1987.

[Map78]
Mapp, T.E. "Applicability of the Rayleigh Curve to the SEL Environment". Unpublished paper, *University of Maryland,* (1978), pp. 1-19.

[McC76]
McCabe, T.J. "A Complexity Measure". *IEEE Transactions on Software Engineering,* Vol. SE-2, No. 12, (December 1976), pp. 308-20.

[McC78]
McCue, G.M. "IBM's Santa Teresa Laboratory - An Architectural Environment for Program Development". *IBM Systems Journal.* Vol. 17, No. 1, (1978), pp. 4-25.

[McC81]
McCracken, D.D., and Jackson, M.A. *Systems Analysis and Design - A Foundation for the 1980's,* edited by: M.M Cotterman et al., (1981), pp. 551-553. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti, W.W., pp. 23-26. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[McC89]
McCabe, T.J., and Butler, C.W. "Design Complexity Measurement and Testing". *Communications of the ACM,* Vol. 32, No. 12, (December 1989), pp. 1415-1425.

[McCl78]
McClure, C.L. *Reducing COBOL Complexity Through Structured Programming.* New York: Van Nostrand Reinhold, 1978.

[McG80]
McGowan, C.L. and McHenry, R.C. "Software Management". *Research Directions in Software Technology,* edited by: Wegner, P., pp. 207-253. Cambridge, MA.: The MIT Press, 1980.

[MCS87]
"Using ESTIMACS - The ESTIMACS questions Explained", (draft). *Management and Computer Services, Inc.,* (March 1987).

[Mil56]
Miller, G.A. "The magic number seven, plus or minus two". *Psychological Review,* No. 63, (1956), pp. 81-97.

[Mil80]
Mills, H.D. "The Management of Software Engineering. Part I: Principles of

Software Engineering". *IBM System Journal*, Vol. 19, No. 4, (1980), pp. 415-420.

[Miy85]
Miyazaki, Y., and Mori, K. "COCOMO evaluation and Tailoring" *Proceedings of the 7 th International Conference on Software Engineering,* (1985), pp. 292-299.

[Moh79]
Mohanty, S.N. "Models and Measurements for Quality Assessment of Software". *ACM Computing Surveys,* 11, (1979), pp. 251-275.

[Moh81]
Mohanty, S.N. "Software Cost Estimation: Present and Future". *Software Practice and Experience.* Vol. 11, (1981), pp. 103-121.

[Mye76]
Myers, G.J. *Software Reliability*. New York, Wiley, 1976.

[Mye77]
Myers, G.J. "An Extension to the Cyclomatic Measure of Program Complexity", *SIGPLAN Notices,* (October 1977).

[Mye78]
Myers, G.J. *Composite/Structure Design*, Van Nostrad, New York, (1978).

[Mus83]
Musa, J.D. "Stimulating Software Engineering Progress - A Report of the Software Engineering Planning Group". *ACM Software Engineering Notes* Vol. 8, No. 2, (April 1983), pp. 29-54.

[Nae88]
Naef, R.E.L. "CEACSIZE Software Sizing Model". *Proceedings of the International Society of Parametric Analysis,* Vol. VII, No. 1 (July 1988), pp. 555-589.

[Naj88]
Najberg, A.C. "Application of Expert System Techniques for Estimating by Analogy". *Proceedings of the International Society of Parametric Analysis,* Vol. VII No.1, (1988), pp. 591-614.

[Nau69]
Nauer, P., and Randell, B. *Software Engineering - Report on a Conference Sponsored by NATO Science Committee,* Garmisch, 1968. Scientific Affairs Division, NATO, Brussels 39, (1969).

[Nel66]
Nelson, E.A. "Management Handbook for the Estimation of Computer Programming Costs". *AD-A648750,* Systems Development Corporation. Santa Monica, CA., (October 31, 1966).

[Nor60]
Norden, P.V. "On the anatomy development projects". *I.R.E. Transaction, P.G.E.M.,* Vol. EM-7, No. 1, (1960), pp. 4.

[Nor63]
Norden, P.V. "Useful Tools for Project Management". *Operational Research in Research and Development*, edited by: Dean, B.V. John Wiley & Sons, Inc., (1963).

[Ost87]
Osterweil, L. "Software Processes are Software Too". *Proceedings of the 9th International Conference on Software Engineering,* Monterey, CA.: IEEE, (1987), pp. 2-13.

[Par57]
Parkinson, G.N. *Parkinson's Law and others. Studies in Administration.* Boston, MA.: Houghton-Mifflin, 1957.

[Par72]
Parnas, D.L. "On Criteria to Be Used in Decomposing Systems into Modules".

CACM, Vol. 14 No.1, (April 1972), pp. 221-227.

[Par80]
Parr, F.N. "An Alternative to the Rayleigh Curve Model for Software Development Effort". *IEEE Transactions on Software Engineering,* Vol. SE-6, No. 3 (May 1980), pp. 291-96.

[Par83]
Partsch, H., and Steinbruggen, R. "Program Transformation Systems". *Computing Surveys,* Vol. 15, No. 3, (September 1983), pp. 199-236. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti W.W., pp. 189-226. IEEE catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Par88]
Park, R.E. "The Central Equations of the Price Software Cost Model". *Proceedings of the Fourth COCOMO Users' Group.* Pittsburgh, PN., (1988).

[Par89]
Park,. R.E. "An Open Letter to Cost Model Evaluators". *Journal of Parametrics,* Vol. IX, No. 2, (June 1989), pp. 1-5

[Pet81]
Peters, L. J. *Software Design: Methods and Techniques.* New York: Yourdon Press, 1981.

[Per83]
Perlis, A.J. "Controlling Software Development Through the Life Cycle Model". Printed in: *Software Metrics: An Analysis and Evaluation,* edited by Perlis, A., Sayward F.G., and Shaw, M., pp. 95-110. Cambridge, MA.: The MIT Press, 1983.

[Phi87]
Phillips, L.D. "Requisite Decision Modelling for Technological Projects". Printed in: *Social Decision Methodology for Technological Projects*, edited by: Vlek, C., and

Cvetkovitch G. North Holland, (1987).

[Pfl89]
Pfleeger, S.L. and Bollinger, T. "A Reuse-Oriented Survey of Software Cost Models", unpublished draft, Contel Technology Centre, Fairfax, VA., (June, 1989).

[Pre87]
Pressman, R. *Software Engineering - A Practitioner's Approach.* New York: McGraw-Hill, 1987.

[Pre84]
Prell, E.M., and Sheng, A.P. "Building Quality and Productivity into a Large Software System". *IEEE Software,* (July 1984), pp. 47-54.

[Pri88]
Price Waterhouse International Computer Opinion Survey, (June 1988).

[Put75]
Putnam, L.H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem". *IEEE Transactions on Software Engineering*, Vol. 1, No.2, (1975).

[Put78]
Putnam, L.H. "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System". *Proceedings of COMPSAC '78.* New York: IEEE, (1978).

[Put79]
Putnam, L.H., and Fitzsimmons, A. "Estimating Software Costs". *Datamation*, (September 1979), pp. 189-198.

[Put80]
Putnam, L.H. *Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers.* IEEE Catalog No. EHO 165-1. New York: IEEE, 1980.

[Put80a]
Putnam, L.H. "The Real Economics of Software Development". *Symposium on the Economics of Information Processing*, (December 1980).

[Put81]
Putnam, L.H. *"SLIM* A Quantitative Tool for Software Cost and Schedule Estimation". *Proceedings of the NBS/IEEE/ACM Software Tool Fair.* San Diego, CA., (March 1981), pp. 49-57.

[Put83]
Putnam, L.H., Putnam D.T., and Thayer L.P. "A Method to Measure the 'Effective Productivity' in Building Software Systems". *Proceedings of the International Society of Parametric Analysts*, Vol. 2, No. 1, (April 1983), pp. 95-143.

[Put84]
Putnam, L.H., Putnam D.T., and Thayer L.P. "A Tool for Planning Software projects". *The Journal of Systems and Software*, 5, (January 1984), pp. 147-154.

[Put84a]
Putnam, L.H., and Putnam D.T. "A Data Verification of the Software Fourth Power Trade-Off Law". *Proceedings of the International Society of Parametric Analysis,* Vol. 3, No. 1, (May 1984), pp. 443-471.

[Put87]
Putnam, L.H. *Quantitative Software Management*, Inc. Size Planner Manual, 1987.

[Rei79]
Reifer, D.J. Tutorial: Software Management. IEEE Computer Society, (1980).

[Rei86]
Reifer, D.J. "Predicting the Size of Real-Time Systems". *Proceedings of the NIS Software Cost and Quality Management Conference* (October 1986), pp. 29-30.

[Rei87]
Reifer, D.J. "Analytical Software Size Estimation Tool - Real - Time *(ASSET-R*: An Overview", RCI-Tn-118, (May 1987).

[Rei88]
Reifer, D.J., and Kane, P.T. "How to Conduct a Software Project Post Mortem". Reifer Consultants Inc. Los Anglos, CA., (1988).

[Rob88]
Robb, F.F. *"The Management of Complexity - Keynote Paper".* Transactions of The Institute of Measurement and Control, Vol. 10, No. 3, (August 1988), pp. 116-121.

[Rob89]
Robb, F.F. "Cybernetics and Superahuman Autopoietic Systems". *Systems Practice* Vol. 2, No. 1, (March 1989), pp. 47-74.

[Roy70]
Royce, W.W. "Managing the Development of Large Software Systems". *Proceedings of the 9th International Conference on Software Engineering,* (1987), pp. 328-338.

[Rub83]
Rubin, H.A. "Macro-Estimation of Software Development Parameters: The Estimacs System". *IEEE.* Catalog No. CH1919-0, (1983), pp. 109-118.

[Rub85]
Rubin, H.A. "A Comparison of Cost Estimation Tools. (A Panel Session)". *Proceedings of the 8th International Conference on Software Engineering. IEEE,* (1985), pp. 1-3.

[Rub85a]
Rubin, H.A. "The Art and Science Of Software Estimation: Fifth Generation Estimators". *Journal of Parametrics.* Vol. 5, No. 2, (June 1985), pp. 50-65.

[Saa80]
Saaty, T.L. *The analytical Hierarchy process.* McGraw-Hill, 1980.

[Sch77]
Schutt, D. "On a Hypergraph Oriented Measure for Applied Computer Science". Printed in: *Proceedings of COMPCON'77*, New York: IEEE, (1977).

[Sch83]
Scharer, L. "The Prototyping Alternative". *ITT Programming.* Vol. 1, No. 1, (1983), pp. 34-43. Reprinted in: *Tutorial on New Paradigms for Software Development.* edited by: Agresti. W.W., pp. 59-68. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[She79]
Sheppard, S., B. Curtis, P. Milliman, and T. Love. "Modern Coding Practices and Programming Performances". *IEEE Computer,* Vol. 12, (1979), pp. 41-49.

[She80]
Shen, V.Y., and H.E. Dunsmore. "A Software Science Analysis of COBOL Programs". *CSD-TR-348,* Department of Computer Science, Purdue University (August 1980).

[She82]
Shen, V.Y., Conte S.D., and Dunsmore H.E. "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support". Unpublished paper, (1982).

[She83]
Shen, V.Y. "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support", *IEEE Transactions on Software Engineering,* SE-9 (2), (1983), pp. 155-156.

[She88]
Shepperd, H. "A Critique on Cyclomatic Complexity as a Software Metric".

Software Engineering Journal, (March 88), pp. 30-36.

[Sho83]
Shooman, M.L. *Software Engineering.* New-York: McGraw-Hill, 1983.

[Sho79]
Shooman, M.L. "Tutorial on Software Cost Models". *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art.* IEEE Catalog No. TH0067-9. New York: Institute of Electrical and Electronics Engineers, (1979), pp. 1-19.

[Som85]
Sommerville, I. *Software Engineering*, 2nd Edition. London: Addison Wesley, 1985.

[Ste87]
Steward, D.V. *Software Engineering with System Analysis and Design,* Brooks/Cole Publishing Company, 1987.

[Str66]
Stroud, J.M. "The Fine Structure of Psychological Time". *Annals of the New York Academy of Science,* (1966), pp. 623-631.

[Swa82]
Swartout, W., and Balzar, R. "On the Inevitable Interwining of Specification and Implementation". *Communications of the ACM*, (July 1982), pp. 438-440. Also in: edited by: Agresti, W. W., pp. 29-37. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Sym88]
Symons, C.R. "Function Point Analysis, Difficulties and Improvements". *IEEE Transaction on Software Engineering,* Vol. SE-14, No.1, (January 1988).

[Tau83]
Tausworthe, R.C. "Software Economics. Cost and Schedule Forecasting". *Workshop on Estimating Software Costs.* Organised by Hyman Silver & Assoc. Ltd. Tel-Aviv, (1983).

[Tha84]
Thadhani, A.J. "Factors Affecting Productivity During Application Development". *IBM Systems Journal* 23, (1984), pp. 19-35.

[Tha88]
Thayer, L.P. "Productivity Improvement Trends - A Quantified Result" *Proceedings of the International Society of Parametric Analysis* Vol. VII, 1 (July 1988), pp. 771-808.

[Ton79]
Tonies, C.C. "Project Management Fundamentals". In: *Software Engineering.* edited by: Jensen, R.W., and Tonies C.C., pp. 553-567. Prentice-Hall Publications, (1979).

[Usg79]
Usg79. *Analysis of nine US federal Projects*, [FG-MSD-80-4]. United States General Accounting Office, (1979).

[Ver87]
Verner, J.M., and Tate G. "A Model for Software Sizing". *The Journal of Systems and Software*, No. 7, (1987), pp. 173-177.

[Ver89]
Verner, J.M., Tate, G., Jeckson, B., and Hayward, R.G. "Technology Dependence in Function Point Analysis: A Case Study and Critical Review". *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, (May 1989), pp. 375-382.

[Wal77]
Walston, C.E., and Felix, C.P. "A Method for Programming Measurement and Estimation". *IBM Systems Journal,* Vol. 16, No. 1 (January 1977), pp. 54-73. Reprinted in: *Writings of the Revolution: Selected Readings on Software Engineering,* edited by: E. Yourdon, pp. 389-408. New York: Yourdon Press, 1982.

[Wal79]
Walston, C.E. "Working Group on Software Cost". Printed in: *Workshop on Quantitative Software Models for Reliability, Complexity, and Cost: An Assessment of the State of the Art.* IEEE Catalog No. TH0067-9. New York: IEEE, (1979), pp. 240-242.

[War83]
Warburton, R.D.H. "Managing and Predicting the Costs of Real-Time Software". *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 5, (September 1983), pp. 562-569.

[Web79]
Webster's new collegiate dictionary. G. & C. Merriam Company, Springfield, MA.: USA, 1979.

[Weg80]
Wegner, P. "Introduction to Part II. - Software Methodology". Printed in: *Research Direction in Software Technology*, edited by: P. Wegner., pp. 203 - 206. Cambridge, MA.: The MIT Press, 1980.

[Weg80a]
Wegner, P. "Introduction and Overview". Printed in: *Research direction in software technology*, edited by: P. Wegner., pp. 1-36. Cambridge, MA.: The MIT Press, 1980.

[Wei74]
Weissman, L.M. "A Methodology for Studying the Psychological Complexity of Computer Programs". *Ph.D. Dissertation. Technical Report CSRG-37. Computer System Group M551A4.* University of Toronto, (August 1974).

[Wei74a]
Weissman, L. "Psychological Complexity of Computer Programs: An Experimental Methodology". *ACM SIGPLAN Notices* 9, 6, (June 1974), pp. 25-36.

[Wei84]
Wiener-Ehrlich, W.K., Hamrick, J.R., and Rupolo, V. F. "Modelling Software Behaviour in Terms of a Formal Life Cycle Curve: Implication for software Maintenance". *IEEE Transactions on Software Engineering,* SE-10 (4), (July, 1984), pp. 376-383.

[Whi90]
Whitley, E.A. "Embedding expert systems in semi-formal domains: Examining the boundaries of the knowledge base". A Doctoral Thesis. University of London, (1990).

[Wil88]
Williams, L.G. "Software Process Modelling". *Proceedings of the 10th International Conference on Software Engineering IEEE*, (1988), 174-186.

[Win82]
Wingfield, C.G. "*USUCSC* Experience with *SLIM*" *Technical Report AWAR 360-5, USA.* Army Institute for Research in Management Information and Computer Science, (1982).

[Win87]
Wingrove, A. "Software Failures are Management Failures". In *Software Reliability: Achievement and Assessment.* B. Littlewood Blackwell Scientific Publications, Oxford, (1987).

[Wol74]
Wolverton, R.W. "The Cost of Developing Large Scale Software". *Software Cost Estimating and Life Cycle Control,* edited by: Putnam, L.H. IEEE Computer Society, (1980).

[Wol84]
Wolverton, R.W. "Software Costing". Printed in: *Handbook of Software Engineering,* edited by: Vick, C.R., and Ramamoorthy, C.V., pp. 469-493. Van Nostrad Reinhold Company. New york, 1984.

[Zav82]
Zave, P. "An Operational Approach to Requirements Specification for Embedded Systems". *IEEE Transactions on Software Engineering.* Vol. SE-8, No. 3, (May 1982), pp. 250-269. Reprinted in: *Tutorial on New Paradigms for Software Development,* edited by: Agresti W.W., pp. 159-178. IEEE Catalog No. EH0245-1. Washington, D.C.: IEEE Computer Society Press, 1986.

[Zav84]
Zave, P. "The Operational Versus the Conventional Approach to Software Development". *Communications of the ACM.* Vol. 27, No. 2, (1984), pp. 104-118.

[Zel80]
Zelkowitz, M.W. "A Case Study in Rapid Prototyping". *Software Pract. Exper.* 10, 12, (December, 1980), pp. 1037-1042.

[Zel82]
Zelkowitz, M.W., and Branstad, M. *Proceedings ACM SIGSOFT Rapid Prototyping Symp.*, Columbia, MD., (April 1982), pp. 19-21.

GLOSSARY OF TERMS

AI	Artificial Intelligence
ASSET-R	Analytical Software Size Estimation Technique
AT&T	American Telephon and Telegrph
Bang	A measure of software size
BT	British Telecom
BYL	Before you Leap
CD	Cost Driver
CEI	Computer Economics, Inc.
CEIS	CEI sizer
COCOMO	COnstructive COst MOdel
COSMOS	Name of an Esprit project
CSF	Critical Success factors
CTC	Corrected Token Count
DBMS	Data-base Management Systems
DI	Degree of Influence
DOD	Department of Defence
DSI	Delivered Source Instruction
DLOC	Delivered lines of Code
EEM	Effort Estimation Model
ESD	Electronic System Division
ESPRIT	Europeam Strategic Research Programe of Information Technology
ESTIMACS	An estimation tool based on FP technique
FP	Function points
FPA	Function point Analysis
FPR	Function Primitive
FPV	Function Point Value
FSP	Full time equivalent Software Ppersonnel

4GL	Fourth Generation Language
IBM	International Business Machine
ICL	Imperial Computers Limited
IIT	IIT Research Institute
IS	Information Systems
ISPA	International Society for Parametric Analysis
IT	Information Technology
JS-2	Jensen's cost estimation model
KDSI	Thousands Delivered Source Instruction
KLOC	Thousands Lines of Code
LOC	Lines of Code
MERMAID	Name of an Esprit Project
MIS	Management Information System
MMRE	Mean Magnitude of Relative Error
MPSS	Most Productive Scale Size
MRE	Magnitude of Relative Error
NASA	National Aeronautics and Space Administration, US
OB	Objects in the automated part of the data-model
P-count	Counts of primitive
PH	Person Hours
PIMS	Project Integrated Management Systems
PM	Person Months
PY	Person Years
PERT	Program Estimating and Reporting Tool
PP	Project planning phase
PRED(l)	PREDiction level

PSD	Preliminary Systems Design phase
RCI	Reifer Consulants, Inc.
RE	Inter-object RElationships in the automated part of the data-model
REP	Request for Proposal
RSC	Relative Structural Complexity
SDLC	Software Development Life Cycle
SDC	System Development Corporation
SEER	Software Estimation and Evaluation Resources System
SLOC	Source Lines of Code
SLIM	Software Life Cycle Methodology
SOFTCOST	A composite estimating model, developed at Jet Propulsion Laboratory, under the direction of Tausworthe.
SPEM	Software Productivity Evaluation Model
SPQR	Softwware productivity Quality and Reliability
SSA	Software Sizing Analyser
SSM	Software Sizing Model
TC	Token Counts.
TCF	Technical Complexity Factor
UFP	Unajusted Function Point
WBS	Work Break Structure
QSM	Quantitative Software Management, Inc.
QSM(SC)	Quantitative Software Methods, Standard Components Sizing.
QSM(FL)	Quantitative Software Methods, Fussy Logic sizing