SOFTWARE DEVELOPMENT MANAGEMENT USING METAMODELS AND ACTIVITY NETWORKS

by

Christian Walker Dawson

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology

15 June 1994

For Sarah

ABSTRACT

This thesis develops the concept, management and control of *metamodels* for the management of software development projects. Metamodels provide a more flexible approach for managing and controlling the software engineering process and are based on the integration of several software development paradigms. Generalised Activity Networks are used to provide the more powerful planning techniques required for managing metamodels. In this thesis, both new node logics, that clarify previous work in this field, and Generalised Activity-on-the-Arrow and Generalised Activity-on-the-Node representations are developed and defined. Activity-on-the-Node representations reflect the current mood of the project management industry and allow constraints to be applied directly to logical dependencies between activities. The Generalised Activity Networks defined within this thesis can be used as tools to manage risks and uncertainties in both software developments and general engineering projects. They reflect the variation and uncertainties in projects more realistically and improve the planning and scheduling of such projects.

An improved Monte Carlo simulation, that allows the number of simulations to be determined dynamically, is used in the temporal analysis of both Generalised and Probabilistic activity networks. The affect that various discrete and continuous activity temporal functions have on the duration of activity networks of different sizes and complexities is also examined. The results of this work, and the comparative simulation requirements of Generalised and Probabilistic Activity Networks, are presented.

These three areas are tied together by a common thread that runs through the main text of this thesis. This thesis provides a new software development modelling concept (metamodels), a technique to support the management of this concept (Generalised Activity Networks), and develops a means of analysis for this technique. These developments are directed at the project management of software development rather than the embedded design processes that are more the concern of the systems analyst.

By considering the software development process from several aspects, specific artificial intelligence techniques can be applied to particular aspects of that process. This thesis investigates how blackboard technology can be used as a framework on which an artificial intelligence support element can be developed. This support assists decision making during particular phases of the software development process. Reason maintenance is employed to allow alternative solutions to the software development process to be evaluated concurrently by allowing several plans to coexist at different levels of a blackboard structure.

KEYWORDS

Software Development, Metamodels, Project Management, Activity Networks, Monte Carlo Simulation, Blackboard Structure, Reason Maintenance

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Ray Dawson, for his guidance and advice over the years that have le d to this thesis being written.

I am also grateful to the University of Derby, not only for practical support but also for the help and encouragement provided by many of my colleagues there.

This work would not have been possible without support from many friends, colleagues, and members of my family. To all these people, who are too numerous to mention, my grateful thanks.

Most of all I would like to thank my wife, Sarah, who has encouraged me throughout my work and been a great support.

Chris Dawson

June 1994

CONTENTS

Chapter 1

Introduction, Aims and Objectives

Chapter 2

Development of Metamodels for Managing the Software Development Process

Chapter 3

Generalised Activity Networks for Project Management

Chapter 4

Project Management and Activity Networks

Chapter 5

Temporal Analysis of Activity Networks

Chapter 6

An Artificial Intelligence Approach to Software Development Management

Chapter 7

Summary, Evaluation, Conclusions

CHAPTER 1

Introduction, Scope, Aims and Objectives

CHAPTER PREFACE

This chapter puts the work of this thesis into an overall context. It begins by introducing the broader field of project-based management before looking in more detail at a subset of this area - software development management - and techniques for improving this activity. It looks at previous work in the field and identifies areas to which this thesis makes particular contributions. The evolution of activity networks is detailed and current gaps in these techniques are identified. This thesis aims to fill these gaps, and looks more closely at how flexible approaches can be used to model the software development process.

CHAPTER KEYWORDS

Software Development, Project Management, Activity Networks

1.1 INTRODUCTION

'. . . a software product is a model of the real world, and the real world is constantly changing' [Schach 1993].

The above quotation encapsulates perhaps the most difficult aspect of software development. That difficulty relates to the inherent changeability of a software product and the environment in which it is developed. This problem makes the development of software systems a particularly complex project area to manage - far more so than that needed for established engineering projects. The problems inherent within the development of software have lead to numerous project overruns and failures over the last thirty years. This has resulted in 'The Software Crisis' that was initially identified by a NATO study group as long ago as 1967. This thesis addresses the problems associated with the management of the software engineering process and develops models and management techniques that overcome some of the difficulties involved.

This thesis approaches the software crisis from two directions. First, it identifies the need for more dynamic models that can manage more flexible systems development. The concept of metamodels, that represent a hybrid of several other models, is developed within chapter two. Second, it looks at one of the techniques of project management - activity networks - and develops theories behind this work. That part of the thesis concentrates on concepts in project management information systems - primarily aimed at the software development process. It concentrates on activity networks, how these techniques can be developed to provide more realistic representations of projects, and improved approaches to their analysis.

1.2 THE SOFTWARE DEVELOPMENT PROCESS

1.2.1 Overview

There are numerous texts devoted to the subject of Software Engineering - for example Schach (1993), Sommerville (1993), Macro (1990), Gilb (1988), Pressman (1994), and Macro and Buxton (1987). All of these provide different models, phase sets, methodologies and so on that aim to elevate some of the problems of a dynamic development, by imposing some form of structure on it.

The main theme behind this thesis is the management of the software development process. There are several aspects to this process - models, methods, activities, support elements and phases that are covered in some detail in chapter two. This section briefly introduces the main stay of this process - software development models - and identifies where this thesis makes advances in this area.

1.2.2 Models

'in order to be able to manage a software project it is essential to follow a defined life-cycle model' [Mazza 1989].

Models are the skeletal structure of the software development process. They provide a visual framework in which this inherently invisible product can be developed. Unfortunately, software development models have proved to be rather restrictive and have never really allowed software to grow within their structure. This has been noted by several authors (for example, Agresti (1986a and 1986b)) and alternative models have been developed over recent years. This thesis develops the concept of hybrid models or metamodels that provide a more flexible structure in which modern day software systems can evolve.

Liu and Horowitz (1989) identified three advantages of using models -

(i) They allow one to understand and explain to others the steps involved in the software process.
(ii) They assist the management of the process.
(iii) They provide a foundation for building tools that enhance the software process.

The first ever model developed was the Stagewise model which is the origin of the widely used Waterfall or Classical Life Cycle model. These models were based on original engineering practices and can be traced back as long ago as 1956 [Benington 1956]. Inadequacies of this and other early models have come to light over the years. According to Turner (1993) the original models 'discouraged effective approaches to software development such as prototyping and software reuse'. Liu and Horowitz (1989) also criticised the Waterfall model for four reasons:

(i) It is foolish to believe that one model is appropriate for all software development projects.
(ii) It provides an inadequate modelling of requirements change.
(iii) It does not involve end users in the process.
(iv) It fails to treat software development as a problem solving process.

Pulk (1990) also pointed out that attempts to strictly adhere to the Waterfall model have been unsuccessful due to scheduling pressures causing overlap of development stages. Overlap between phases in a software development is inherent and desirable within this process. This overlap should not be discouraged.

Due to the inadequacies of the Waterfall model several other models have been developed over the past ten to fifteen years. Some represent hybrids of methodologies and techniques whilst others represent the implementation of methodological approaches. Examples include Prototyping Models [Bowen 1990], Evolutionary Deliveries [Sommerville 1993], the Spiral Model [Boehm 1988], Formal Transformation Models [Sommerville 1993], and 4GL Models [Pressman 1994]. Liu and Horowitz (1989) developed a hybrid model consisting of And/Or graphs and Petri Nets called the DesignNet Model. This model appears more complex than the standard model approach which identifies the stages through which a software development progresses at a strategic level. Models are covered in more detail in chapter two.

In order to put forward their own ideas of a software development model, Liu and Horowitz (1989) identified six idealised features of models:

(i) They must adequately describe software development as a design process.
(ii) They must accept software development as a parallel process - many people doing several tasks simultaneously.
(iii) Activities can be undertaken when their diverse conditions exist - for example, a simple case would involve their resources being available and their preceding activities having completed successfully.
(iv) They should be able to indicate all artifacts that are produced at various points in the process - for example, documentation at each stage.
(v) If an activity fails, they should be able to indicate the activities and resources affected. Affected activities may have to be re-executed.
(vi) They should be able to indicate the extent and nature of resources used by a subtask.

The metamodel, developed within chapter two of this thesis, addresses points (i) and (iv). By supporting this model through the development of Generalised Activity Networks in chapter three, and their subsequent analysis in later chapters, this thesis also supports the other idealised features of a model.

1.3 SOFTWARE DEVELOPMENT MANAGEMENT

1.3.1 Project-Based Management

'to plan and manage a software project successfully, we must view project management as a process, and the project plan as an activity that prepares data for that process' [Rexing 1991].

Since software developments can be viewed, perhaps, as a specific subset of projects in general, it is worth looking first at project-based management. This field is well established - as early as 1963 the US Air Force PERT Orientation and Training Center was able to cite 702 works in this field [Dooley 1964].

According to Turner's work (1993) project-based management involves three integrated dimensions: *objectives*, *management processes*, and *levels*. The management processes (plan, organise, implement, and control) are identifiable with project life cycle phases. The different phase sets applicable to the software development process are discussed in chapter two.

The five objectives identified by Turner are scope, organisation, quality, time and cost. Earlier approaches to project-based management focused on the management of quality,

cost and time objectives. The feeling was, that if these three objectives could be managed, then projects would be completed successfully. However, Turner goes on to state that these objectives are, in fact, optional 'soft constraints'. It is scope and organisation that are obligatory. 'Without scope there is no project; without the organisation it cannot be implemented' [Turner 1993].

All organisations can be viewed as some form of hierarchical structure. Generally there are no more than eight levels within an organisation [Lucey 1987]. Taking a more general stance one can define three levels within an organisation. Turner (1993) defined three project levels as integrative, strategic and tactical. For the purposes of this thesis this terminology will be maintained. The objectives of a project must lie within an organisation's objectives which are represented by the integrative level (see chapter four). At the strategic level 'a strategy for achieving the purpose is defined' [Turner 1993]. This strategy is viewed in this thesis as the model level of the software development process. The tactical level then represents ways of achieving these strategic targets. Generalised Activity Networks, defined in chapter three, can be used at both the strategic and tactical levels.

1.3.2 Software Development Management

'Most software developers regard keeping pace with ever-changing user requirements as their main challenge of the moment' [Peltu 1994].

Software developments represent a special kind of project. Unlike most engineering projects they prove to be particularly awkward to manage as their development processes are generally ill-defined and dynamic. One way these problems are overcome is to define a model that represents the way in which software is developed. To go from a set of requirements to a working system has to imply a certain process anyway. It was the lack of an applicable process in the late 1960s that initially lead to the software crisis. These days, however, even though several new models exist, software is still being delivered late and with errors. Personal contact with several software houses highlighted reasons for these problems - although new models exist, most developers use traditional Life Cycle approaches or, in some cases, no models or methodologies at all. They also tend to use more traditional project-based management approaches without regard to software's inherently dynamic nature.

Even during the 1980s authors were still looking at how wider project management approaches could be applied to the software development industry. Blaney (1989) was one such author who stated 'There is no reason that project management techniques applied successfully in other industries cannot be applied in the software development

industry'. Tulip (1983) proposed that project management techniques used in the construction industry and contract management are equally applicable to the management of data processing projects. Tausworthe (1980) also applied general project management techniques from engineering to software project management, as did Carter, Clare and Thorogood (1987).

The problem overlooked by these authors is that software development is inherently different to general engineering projects and requires a more knowledgeable approach. Although project management skills are required, more knowledge of the product is needed. Indeed, Pulk (1990) stated that a software project manager needs all of the following to be an effective leader:

(i) Project management skills.
(ii) Software development skills.
(iii) A knowledge of the product being developed.

In order to assist the problems faced by the software development industry, project management techniques need adopting and adapting. One of the techniques, taken from the broader field of project management, is that of project management information systems. Although these systems provide some means of support they need development to include risk and probabilistic analysis. Blaney (1989) confirmed this idea when he identified that one of the unique issues that drives project management in the software development industry is that 'Project management software that supports probabilistic risk analysis must be used to predict realistic completion dates'.

1.4 PROJECT MANAGEMENT INFORMATION SYSTEMS

'Tools are too inflexible and don't do what I want . . . . they seem to have been written by programmers not project managers' cited in Peltu (1994).

Project management information systems provide computer-based support for project managers. The most popular of these systems are networking systems that are based on project management techniques dating back to the late 1950s (PERT and CPM) and earlier (Gantt charts). Not only do these systems provide a means of project planning but they also assist with project control. According to Turner (1993) three requirements of a project management information system include -

(i) Integration across an organisation.
(ii) Planning and control.

(iii) Fast response.

It is the last two of these requirements that are addressed by developments within this thesis. Improved planning and control is covered by developments within chapters two and three, and a faster response is dealt with by developments in chapter four.

Although there was a plethora of research in the 1960s on activity networks, this tailed off somewhat during the 1970s and 1980s. It has only been recently revived in the new information explosion of project management software tools (network based) that have appeared over recent years. Jacobs (1994) identified this by noting that 'the specialised nature of the topic and the non-visual nature of PCs meant that further development of the programs already produced took a back seat for a number of years'. The late 1970s and 1980s is perhaps better remembered for its development of models and methodologies, in the field of software development management, than for project management. It is not worth denying, however, that project management software tools today are particularly user friendly and take much of the work out of the analytical side of project management. They provide comprehensive reporting facilities for all management levels and a means of planning, calculating and controlling all sizes of project. Appendix A provides a good visual example of one such tool in use. This appendix shows reports (various work breakdown structures, activity networks and Gantt charts) generated for the Milltown Road bridge project by CA-SuperProject®. This is one of the more popular PC-based project management software tools on the market today, and provides, like many of its rivals, an invaluable aid to the project manager. The only drawback with these tools is that they are still based primarily on techniques developed during the late 1950s. It is only through the use of more powerful Windows™ based machines that these techniques have become more usable. Now that the processing power is available, more radical developments are needed to take these project management tools into the next century.

1.5 ACTIVITY NETWORKS

1.5.1 Overview

The earliest approach to project representation is the Gantt chart (after Henry L Gantt) or Bar chart. This approach is still in widespread use today and almost all project management information systems (network based) incorporate some means of representing these charts. Examples of Gantt charts can be found in Appendix A (pages A3 to A8). Since the 1950s, Gantt charts have been complemented by Activity Networks that allow the relationships between activities in a project to be explicitly

identified.

Since their inception, Activity Networks have become invaluable aids to the planning and management of a multitude of projects throughout business and industry. Activity Networks originated in two very similar forms in the late 1950s: PERT (Program Evaluation and Review Technique), and CPM (the Critical Path Method). CPM is sometimes referred to as CPA (Critical Path Analysis) but this relates more to the subsequent analysis of these networks than their actual representation.

Both PERT and CPM have become intertwined over the years to form the basis of networking techniques that are used today. Generally speaking, either term is now used to refer the approach of representing a project by a network diagram, performing various calculations to determine a project's duration, resource requirements, and costs, and controlling a project through this medium.

The original PERT and CPM techniques represented the activities of a project by arrows that connected nodes or events. Consequently these techniques are sometimes referred to as Activity-on-the-Arrow (AoA) or the Arrow Diagram Method. In Activity-on-the-Arrow Networks, dummy activities (dashed arrows) are needed to define some of the interrelationships between tasks. These dummy activities have zero cost and time parameters associated with them. An example of a simple Activity-on-the-Arrow network is shown in figure 1.1.

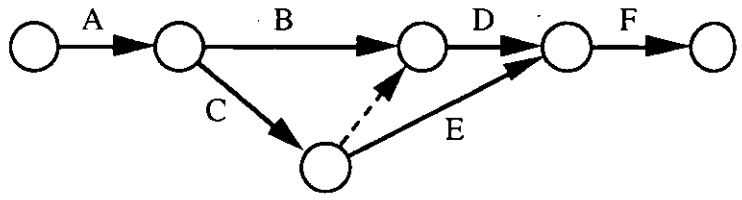


Figure 1.1 A Simple Activity-on-the-Arrow Network

An alternative representation to Activity-on-the-Arrow is when activities are represented by nodes and their relationships are represented by arrows. This alternative representation is called Activity-on-the-Node (AoN) and is logically equivalent to an Activity-on-the-Arrow representation. An example of an equivalent simple Activity-on-the-Node network is shown in figure 1.2.

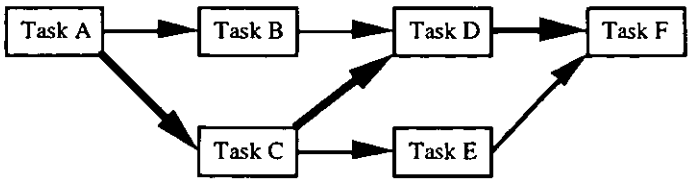


Figure 1.2 A Simple Activity-on-the-Node Network

The first Activity-on-the-Node representation was developed in 1958 by Fondahl at Stanford University [Levine 1986]. To many, an Activity-on-the-Node representation provides a clearer representation of a project as it does away with the need for dummy activities that can complicate a project plan. Consequently Activity-on-the-Node is the more popular of the two representations in use today. The advantages of Activity-on-the-Node are discussed in chapter three where a Generalised Activity-on-the-Node representation is developed.

Activity Networks can be split into three categories of complexity according to their activity characteristics - Deterministic, Probabilistic and Generalised Activity Networks. CPM falls into the simplest of these three categories (Deterministic) and PERT into the Probabilistic category.

1.5.2 Critical Path Method

The Critical Path Method was developed in 1957 by the DuPont Company and Remington Rand for use in the construction industry. As activities within the construction industry are generally well understood and have been performed several times before, previous experience can be used to predict activity durations and costs accurately. In the original CPM approach two time estimates were required for the duration and cost of each activity - called *normal* and *crash*. The normal time represents the duration to complete a project requiring the least amount of money. The crash time and cost represent the minimum possible time to complete a project with an associated increase in cost. Consequently CPM can calculate an estimate of the most economical or shortest time to complete a project. Because of the simple representation of activity times and costs in CPM, it falls into the Deterministic Activity Network category.

Deterministic Activity Networks are used to manage projects in which the activities of a project are well understood and complete in recognised times. Well established algorithms (for example Moder and Phillips (1983), and Whitehouse (1973)), that make forward and backward passes through an activity network, are used to calculate various

data relating to a project's times and costs in these networks. These data include information such as the early start, late start, early finish and late finish of each activity, the critical path and several different slack or float times. Appendix B provides formulae for these calculations.

1.5.3 Program Evaluation and Review Technique

PERT was developed in 1958 by the Lockheed Missile and Space Division, the United States Navy and consultants from Booz-Allen and Hamilton Company. It was developed to control the development of the Polaris Fleet Ballistic Missile.

As identified earlier, PERT is a Probabilistic Activity Network representation. Probabilistic Activity Networks help to manage projects where there is some uncertainty about the duration of activities in a project. This is usually the case for new projects where activities have not been performed previously and an exact duration cannot be estimated accurately. In these situations the activity durations are represented by probability distribution functions (temporal functions). Due to the stochastic nature of the activity times in Probabilistic Activity Networks, they tend to be difficult to analyse. A way of analysing both Probabilistic Activity Networks and Generalised Activity Networks will be studied in chapter four.

PERT has developed through four generations over the years; PERT/Time, PERT/Cost, PERT/LoB (Line of Balance) and PERT/LoB/Cost. PERT/Time is the earliest and most basic version of PERT while the other techniques include enhancements to the simple PERT/Time idea.

(i) Second Generation: PERT/Cost

PERT/Cost was issued as a set of guidelines in 1962 by the American Department of Defense and NASA. It was developed for the specific purpose of integrating time data with associated financial data of physical accomplishment. It established certain cost tracking parameters as requirements for selected DOD and NASA projects. These parameters had to be integrated into the project schedule.

(ii) Third Generation: PERT/LoB

In 1967 Schoderbek and Digman [Schoderbek and Digman 1967] developed the third generation of PERT - PERT/LoB. The LoB technique had been used as an effective management tool in the control of steady state production activities for twenty five years prior to its amalgamation with PERT. The PERT/LoB technique aimed to bring together the development techniques of PERT, with the production techniques of LoB. PERT/LoB covered the critical transition phase between these two phases.

(iii) Fourth Generation: PERT/LoB/Cost

This system encompasses time, cost, and resource scheduling. One published PERT/LoB/Cost system is Cost and Schedule Planning and Control or CSPC [Saitow 1969]. This method not only covers control from a project managers viewpoint, but it also provides reporting facilities for upper level management.

1.5.4 Resource Allocation

It was not until the mid-1960s that project management techniques began to take into consideration the fact that resources (staff, machinery, money etc) are seldom available to a project in unlimited quantities. Three possible methods of handling limited resources have been developed over the years [Guerrieri 1987]:

(i) Resource Indication - this system highlights those areas where scheduled resources exceed available resources - it does not solve the problem but merely identifies it.

(ii) Resource Levelling - reduces the amount of variability of resource usage over the project duration when sufficient resources are available, and the project must complete in a given time. This can be achieved by various methods including shifting non-critical tasks, splitting activities and so on.

(iii) Fixed Resource Limits Scheduling - this technique has essentially the opposite constraint to resource levelling. In this technique the resource limits are fixed, but the project completion date is allowed to slip within given constraints. Resource limits can be fixed over the project duration or allowed to vary between certain limits.

Combinations of these techniques are feasible and depend on the network type and the way in which activity durations are represented. It should also be possible to assign priorities to activities that are to be levelled. In this situation lower priority activities will be stretched/delayed first ensuring project priorities are maintained. There are numerous algorithms available for resource levelling and fixed resource limits scheduling - earlier publications describing different algorithms include Wiest (1967), Clark (1961a), Levy et alia (1962), King (1964), and Berman (1964).

In the 1960s it was only the large mainframe computer programs that could perform these calculations. One example is RAMPS (Resource Allocation and Multi-Project Scheduling) [Lambourn 1963]. These days even some of the cheaper PC-based computer packages have facilities for resource optimisation. Examples include Project Scheduler™ 6, CA-SuperProject®, Primavera Project Planner®, and CS Project™ that

has its own scheduling criteria called CARLO - Cost and Resource Levelling Optimisation (see chapter four).

1.5.5 Precedence Diagram Method

The Precedence Diagram Method (PDM) was the next real stage in activity network representation. Before PDM was developed the relationships between tasks in activity networks were fixed. In other words, an activity would be deemed to start immediately after its preceding activity(ies) had completed. It was obvious that in many situations this is not the case, and a solution to this problem was required. As an example, take the hardening of concrete foundations in a house building project. There is clearly some delay between pouring concrete to form a foundation and it becoming hard enough for the next stage of the project to commence. In 1973 Crandall [Crandall 1973, Wiest 1981] introduced the Precedence Diagram Method that overcame this problem. PDM is based on Activity-on-the-Node as this provides a more clear representation of the constraints that can be applied. In Activity-on-the-Arrow networks, these constraints require additional nodes and dummy activities that can overly complicate project plans. PDM allows both overlap and underlap between connected activities using a delay factor called lag. A similar, more limited method, called the Metra Potential Model, was developed in 1983 by Gotthardt and Winkelmann [Gotthardt and Winkelmann 1983]. The Metra Potential Model defined a minimum (z) and maximum time (-w) delay between two linked activities but did not incorporate the full power of all four PDM lag types (figure 1.3).

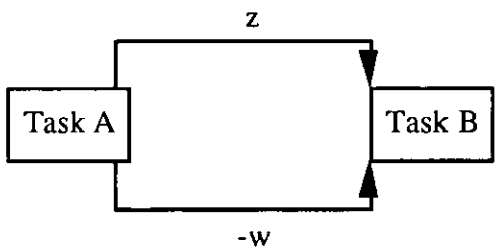


Figure 1.3 The Metra Potential Model

In PDM there are four types of lag that can be defined - Start-to-Start, Start-to-Finish, Finish-to-Finish, and Finish-to-Start (SS-*n*, SF-*n*, FF-*n*, FS-*n* - where *n* represents the lag in associated time units).

- Start-to-Start (SS)

  Task B cannot begin until a delay after the start of task A (figure 1.4). For example,

performance monitoring cannot start until a given time after system implementation has started, to allow time for the system to initially settle down.

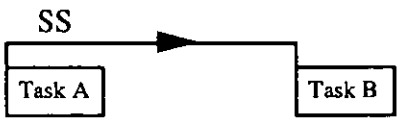


Figure 1.4 Start-to-Start Delay

- Start-to-Finish (SF)

  Task B cannot finish until a delay after the start of task A (figure 1.5). For example, when a new software team starts work there must be some form of hand-over period where the old team cannot finish work until the new team has had time to prepare to take over.

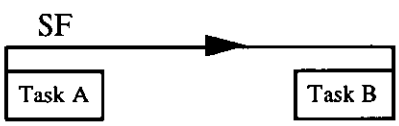


Figure 1.5 Start-to-Finish Delay

- Finish-to-Finish (FF)

  Task B cannot finish until a delay after the completion of task A (figure 1.6). For example, it may not be possible to finish testing until one month after a diagnostics system has been installed. After this time it might be safe to assume that the system is working.

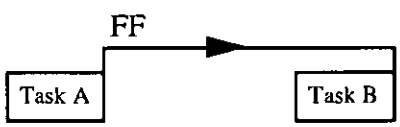


Figure 1.6 Finish-to-Finish Delay

- Finish-to-Start (FS)

  Task B cannot start until a delay after the completion of task A (figure 1.7). For example, if task A represents the giving of notice for a meeting, there will be a delay corresponding to the notification time before the meeting itself takes place - task B.

Figure 1.7 Finish-to-Start Delay

An example of a simple PDM network with all these constraints is shown in figure 1.8. How PDM constraints can be applied to Generalised Activity Networks will be discussed in chapter three.

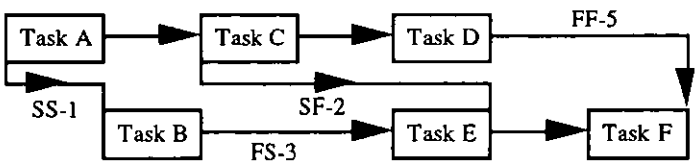


Figure 1.8 A Simple PDM Network

1.5.6 Cost Control

Cost control systems work as a control mechanism within project management information systems. They generally work on an 'earned value' system which works on the principle that subsequent times and costs in a project are affected by times and costs accrued so far. The initial technique for cost control was specified in 1975 by the American Department of Defense when they issued specification DODI7000.2, called *cost/schedule control system criteria* (C/SCSC). At the same time the American Department of Energy had a similar specification called *performance measurement system*, but it is C/SCSC that is still in widespread use today. There are two alternatives to the type of forecasting within cost control systems:

(i) Future work will ensue at the same rates of cost, resource requirements and duration as work completed so far. This means that any variations to the original plan will continue to occur at the same rate in the future.

(ii) Future work will ensue at the previously planned rate. This method assumes that the only variation to the project plan was that already monitored in the work achieved so far.

Examples of project management software tools on offer today which include cost control measures based around C/SCSC are InstaPlan™ 5000, Schedule Publisher™ 4.1 for Windows™, Cascade® by MANTIX which is an organisation wide approach, and CA-SuperProject®. Another system that performs earned value analysis is Parade® (1993) by Primavera that takes plans from Primavera Project Planner® and performs the necessary earned value cost control calculations on these plans.

1.5.7 Generalised Activity Networks

Generalised Activity Networks represent the third and final complexity level of Activity Networks. References to Generalised Activity Networks are particularly sparse when compared with PERT and CPM approaches. Examples of work in this field include Bellas and Samli (1973), Interrante and Biegal (1991), Moore and Clayton (1976), Moore and Taylor (1977), Pritsker (1974 and 1979), Pritsker and Happ (1966), Pritsker and Whitehouse (1966), Samli and Bellas (1971), Taylor and Moore (1978), Whitehouse and Pritsker (1969), McGowan (1987), Moeller (1972), Moeller and Digman (1981), Kidd (1990 and 1991).

Generalised Activity Networks originated in 1962 [Eisner 1962] and evolved into a methodology called GERT [Drezner and Pritsker 1966] (Graphical Evaluation and Review Technique). Although this developed into several other Activity-on-the-Arrow forms (for example, Moore and Clayton (1976)) only one other real advancement was made - that of VERT in 1972 [Moeller 1972, Lee et alia 1982, Kidd 1990 and 1991] (Venture Evaluation and Review Technique). Generalised Activity Networks have never become established as a project management technique. Reasons for this and their development is covered in chapter three.

1.5.8 Summary

Activity networks have three levels of complexity and a number of different representations. The complexity levels range from Deterministic and Probabilistic to Generalised Activity Networks. These networks can ultimately be represented in perhaps four different ways - Activity-on-the-Arrow, Activity-on-the-Node, Precedence Diagram Method and a Hybrid. A Hybrid representation refers to a combination of the other three. It is a Hybrid representation that is developed in chapter three.

Table 1.1 summarises the different kinds of activity networks that are available. No one these days tends to refer explicitly to a specific approach (for example, a Probabilistic Activity-on-the-Node Network representation) preferring to group all activity networks under the PERT/CPM umbrella. This thinking is represented by CPM* and PERT* in

table 1.1.

The gaps within this table (Generalised Activity-on-the-Node and Generalised Precedence Diagram Method) are the gaps filled by this thesis. Chapter three develops a Hybrid representation to fill these gaps and provides a means of analysis.

	Representation		
	AoA	AoN	PDM
DANs	CPM	CPM*	CPM*
PANs	PERT	PERT*	PERT*
GANs	GERT/VERT	-	-

Table 1.1 Activity Networks - Representation and Complexity

1.6 FUTURE WORK

1.6.1 Overview

The majority of popular project management information systems are based on established activity network techniques. Developments over recent years have merely included enhancements to the usability of such packages for example, by incorporating the Windows™ platform. Software development management needs more powerful concepts and supporting software to overcome the chronic software crisis. Areas of potential future research, that represent advancements towards a solution to this problem, are identified within several chapters of this thesis. One such long term aim is the development of intelligent project and software development management systems, development of which has only been patchy over the years.

1.6.2 Intelligent Management Systems

'While the responsibility for project decisions should and must lie with project managers, a knowledge based assistant could provide decision support for these and other project management tasks' [Ahmad et alia 1988].

Ahmad et alia (1988) gives an outline of the tasks and methodologies that an artificial intelligence project management tool should incorporate. It is interesting to note that this work speaks of 'a knowledge based assistant' rather than an intelligent machine that would make all managerial decisions. This implicitly identifies that the decision making process lies with project managers themselves, and machines, certainly at this stage, are not sophisticated enough to take over the decision making process themselves.

An example of an intelligent project management system is Callisto [Sathi et alia 1986]. 'The Callisto project was born out of the realisation that the classical approaches to project management do not provide sufficient functionality to manage large engineering projects'. The Callisto project had four goals that encompass various aspects of the project management field:

(i) Activity modelling - to generate a model of activities and their constraints.
(ii) Configuration management - generate a hierarchical product representation, and develop a system to support change.
(iii) Activity scheduling - schedule with various hard and soft constraints and goals that involve dynamic rescheduling, *what if* simulation and heuristics to guard against 'bad' schedules.
(iv) Project control - study and model the status of updating and activity-tracking procedures and the use of managerial heuristics for reporting, focusing and diagnosing problems.

Generalised Activity Networks, developed within this thesis, address some of the points raised in goals (i), (iii) and (iv) above. The interaction between these networks and intelligent management systems represents a particularly interesting area for future research. Although this work is beyond the intended scope of this thesis chapter six discusses a proposed artificial intelligence approach to software development management.

1.7 CONTEXT

'To provide effective support, project management tools should be tailored to the needs of the decision makers, and not vice-versa' [Ahmad et alia 1988].

This quote encapsulates one of the main themes running through this thesis - that of the development of techniques more applicable to the management of the software development process.

Software developments are a subset of projects in general which are managed through several phases (identified more explicitly within software development by models and phase sets). These developments require the management of different objectives at different organisational levels. Metamodels provide a means of managing software developments at a strategic, and subsequent tactical level for organisation, cost and time objectives.

Activity networks represent a particular means of project representation that are used to assist both the planning and control of projects. Currently they are not flexible enough to cope with situations encountered in both software development projects and other, less clear problem domains (for example, research and development). Generalised Activity Networks provide more flexible activity networks for planning and control. They represent a more applicable project management information system and are supported by analytical techniques developed in chapter four.

1.8 AIMS AND OBJECTIVES

1.8.1 Overview

To summarise the goal of this thesis into one aim is a very difficult task, since in the development of this work, a number of different but interrelating tasks were defined and executed. However, this thesis sets out:

(i) To identify and develop models for more flexible management of software systems development.
(ii) To develop Generalised Activity Networks, both Activity-on-the-Arrow, Activity-on-the-Node and, as a consequence, PDM in order to support more flexible software development management models and other project types.
(iii) To improve analysis and data quality of such management support systems.
(iv) To look at the development of an artificially intelligent system for the management of the software development process.

1.8.2 Contribution

Although the above points may initially appear somewhat disjointed, they are brought together by a common thread that indicates the major contribution made by this thesis. This contribution involves the development of an entirely new concept for software engineering management (metamodels in chapter two), supported by a more flexible management technique (developed in chapter three) which in turn is supported by an improved analysis approach (developed and discussed in chapters four and five).

CHAPTER 2

Development of Metamodels for Managing the Software Development Process

CHAPTER PREFACE

Software development paradigms currently used within industry are based on established models of the software life cycle. As technology has evolved and become more accessible, more flexible approaches to the development of software systems have been advocated. Several new models have evolved over the passed ten years based on more powerful software engineering tools and more sophisticated development methodologies. A more responsive approach to the development of software systems should allow the integration of alternative development models to be achieved without requiring changes in the development management structure. This chapter introduces metamodels that allow more flexible management of software systems development (presented in the paper by Dawson and Dawson (1994c)). It also introduces a technique (Generalised Activity Networks from Dawson and Dawson (1994a)) that provides the management support needed by metamodels.

CHAPTER KEYWORDS

Software Development, Paradigms, Metamodels, Generalised Activity Networks, Planning

2.1 INTRODUCTION

'Every project suffers from continuously changing users' requirements' [Peltu 1994].

Ever since the software crisis was identified by a NATO study group in 1967 [Schach 1993], software practitioners have attempted to understand more fully the process by which software is developed. The early attempts at structuring the software development process were based on established engineering practices. It soon became clear, however, that software was inherently different from physically engineered products. The engineering project practices that had been adopted were (and still are) wholly inadequate. Brooks, in his work of 1987 [Brooks 1987], detailed two features of

software that make it particularly difficult to produce - those inherent within the software and those introduced by accident.

The inherent problems identified by Brooks were *complexity*, *conformity*, *changeability*, and *invisibility*. Complexity derives from both the large number of states in which a software system can reside, and the interaction between software elements that increase nonlinearly with this number of elements. Conformity relates to the problems associated with developing software so that it conforms with other software and hardware systems. Software can be produced to run on various platforms and must provide an interface that is compatible with other systems. As software environments are constantly changing (in terms of hardware and problem domains that are encountered), changeability presents another inherent software problem. The last inherent problem identified by Brooks was invisibility. Invisibility identifies the difficulties involved in representing software in a conceptual, diagrammatical form. According to Brooks, software needs to be represented by several, general directed graphs superimposed on one another - for example, control flow, data flow, dependency, time sequence and so on. Brooks went on to state that in spite of simplifying these structures they remain inherently unvisualisable.

These inherent problems are compounded by accidental problems introduced by software developers. These problems represent difficulties that attend the production of software but are not inherent within it [Brooks 1987]. Accidental problems include using individual programs together (causing data compatibility problems), accidental complexity introduced by developers at higher levels (for example, poor data structuring), and slow software response times (for example, batch processing) possibly causing a user to lose track of the minutia and the thread of what s/he was thinking.

It has already been advocated by many authors that a more flexible approach is needed for the development of software systems. As long ago as 1986 [Agresti 1986a] practitioners were calling for an approach that did not impose a rigid phase structure on the development of software systems such as that imposed by the conventional waterfall model (for example, Benington (1956)). What was not addressed with this proposition was how management could cope with such a process. Project managers are reluctant to embark upon ideas that have little structure or visible direction. This culture, alongside the inertia of large organisations, has resulted in a situation where the majority of software projects undertaken today are based on the established life cycle or waterfall model. In order to distance ourselves from the rigid constraints imposed by the waterfall model other development paradigms have been introduced over recent years. These alternative paradigms can still impose a rather rigid set of constraints on the direction of a system's development by defining the specific stages (and the order)

through which a project develops. One way of overcoming this problem is to combine several different paradigms into a hybrid model called a *metamodel*. Metamodels allow software development projects to evolve along more responsive pathways yet still provide management with a structure to which projects can relate. Metamodels can be controlled by using a more flexible management technique that is introduced later in this chapter.

2.2 THE SOFTWARE DEVELOPMENT PROCESS

2.2.1 Overview

Boehm (1981) identified two different types of software developments:

Projects: Software developments for a single, one-off client.
Products: Software development for a multiplicity of unsecured (and possibly unknown) clients.

Although these definitions identify the resultant operating environments of a software product, the use of the word *project* in these definitions is somewhat misleading. There is no reason why a software development project could not involve the development of a system for a broader market, in which case the *project* would be to develop a marketable *product*. From this perspective, all software developments can be viewed as projects of a sort, but what is meant by the term project?

Software developments, when viewed at their broadest level, can appear as unclear systems with rather vague start and end points. After all, when has a software development started - after the initial idea, when coding begins, when a concept document is approved by developers and customers, or when a contract is signed? When is it complete - when the software leaves the development house, after the second version is released, or after three years of maintenance?

This viewpoint contrasts quite significantly with Barnes' [Barnes 1989] definition of a project - 'something which has a beginning and an end' cited by Turner (1993). Turner qualifies this definition by introducing several other project descriptions. He picks out - 'a one-time, unique endeavour by people to do something that has not been done before' [Smith 1985] - as the definition that captures 'the essence of projects'. Although software has already been shown to have a rather unclear beginning and end, Smith's definition encapsulates the essence of software development, be it a product or a project.

In order to conceptualise the aspects of software development projects, various structures have been devised. The software development process was viewed by Rook (1986), and less explicitly by Macro and Buxton (1987), as a three dimensional system. By imposing a structure on the system in this way it is possible to reduce its complexity. The three aspects or dimensions that encompass this system are *support elements*, *activities*, and *phases*. Figure 2.1, adapted from Rook (1986), provides a tighter representation of his ideas. Some of the detail in this figure is inaccurate in that it identifies certain items, for example *Time Management*, as separate entities to *Project Management*. These items are, in fact, subsumed within Project Management and should not be shown separately. This point is covered in more detail in chapter four.

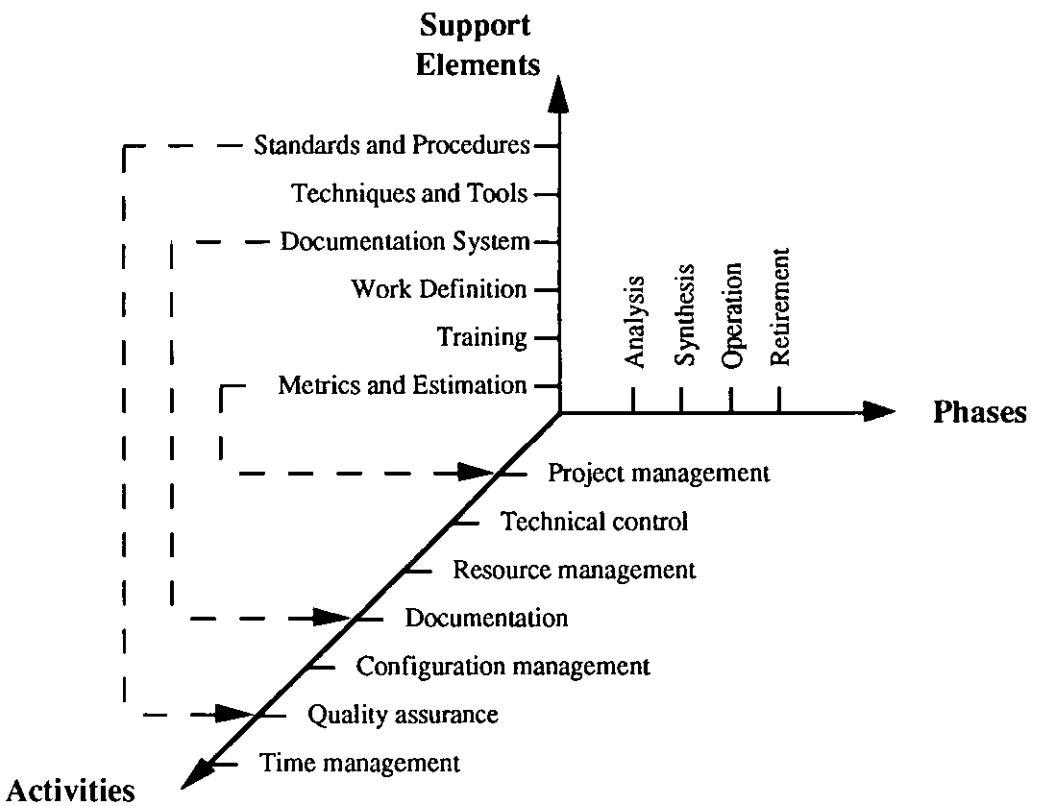


Figure 2.1 Rook's Dimensions

Within Rook's representation there is a close association between certain project activities and specific life cycle phases that have been defined. He identified *requirements specification, structural design, detailed design*, and *code and unit test* as both phases and activities in the software development process. The activities that were defined, however, could represent ongoing tasks that are performed at other stages throughout the life cycle of a project, and should not be restricted to a particular phase. In this representation there is a more apparent link between some of the support

elements and certain activities. The dashed, arrowed lines in figure 2.1 indicate some of these more obvious associations. For example, standards and procedures assist the activity of quality assurance, a documentation system clearly assists project documentation, and metrics can help to simplify the task of project management.

2.2.2 Support elements

Support elements are those tools and procedures that have evolved to assist the software developer. They include aids such as standards and procedures, documentation systems, project management tools [Powell 1990], training, and can be used to assist specific activities or methodologies within particular phases of the project life cycle. CASE tools [Costello 1990, Hill 1990] and IPSEs (Integrated Project Support Environments) [Brown et alia 1986, Brown 1988] fall into this category.

2.2.3 Activities

Activities are ongoing tasks that different members of the software development team perform throughout the lifetime of the project, keeping it running on a daily basis. As noted earlier, some of these activities can be assisted by specific support elements. Several of these activities can also be attributed to specific team members. The project manager, for example, would be responsible for resource and project management activities, and project documentation could well be assigned to a technical author. Activities can also be attributed to particular phases of a software development (although they do not have to be limited to a particular phase). For example, control would, more often than not, be performed during synthesis and planning during analysis.

2.2.4 Phases

Rook originally defined eight phases within his representation. Phases represent the stages through which a project progresses to completion, any number of which could theoretically be defined. Software development phases will be studied in more detail later.

2.2.5 Limitations of Earlier Representations

Unfortunately this structure, identified by Rook, does not represent the software development process in its entirety. Today there are several paradigms available for the development of software systems. By identifying a specific, eight step phase structure within his representation, Rook related his model to a single, waterfall-type development paradigm. He has, in effect, presupposed the software development model and split the

development into a series of steps with specific start, end, and intermediate control points to reflect this. Imposing a specific phase structure on a representation in this way can be particularly dangerous as it can either be so vacuous as to provide no practical value to any software development model, or too specific to relate to alternative models [McCracken and Jackson 1986]. It is possible, however, to identify a set of phases that are broad enough so as not to restrict the process yet detailed enough for the purposes of this chapter. In order to put some of these ideas into context, later in this chapter a broad series of four phases will be defined. However, even with such a broad set of phases, there is always some overlap inherent between each phase. One cannot get away from the fact that phase sets are affected by the development paradigm used. They must not, as is the case with the waterfall model, be so inflexible that the paradigm is restricted within their structure.

2.2.6 A Comprehensive Representation of the Software Development Process

A more comprehensive representation of the software development process is provided in figure 2.2. This augments software development paradigms and methodologies with the three dimensions of Rook's representation.

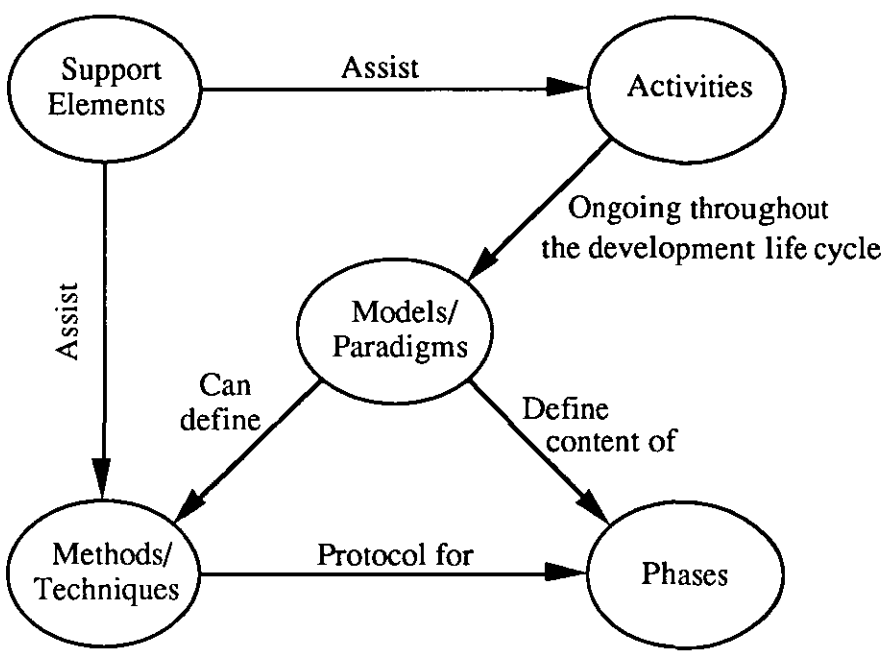


Figure 2.2 Comprehensive Software Development Process Representation

'If you can't model it, you can't build it' [Hill 1990].

Models represent strategic level development plans within an organisation that identify the broad stages through which a software development progresses. Organisational levels and project objectives are covered in more detail in chapter four.

One feature that has been omitted from these representations is that of resources. Although resources can be viewed as external to the process, they do have an influential effect on its progression. The software development process can be viewed, in fact, as a process in which resources are consumed, by going from an initial idea to a fully operational system. Resources are external to the representation of figure 2.2, being consumed by each of the aspects as and when they are required.

This chapter is, therefore, concerned with five aspects that constitute the overall concept of the software development process - Models (also called Paradigms), Methods, Activities, Support Elements, and Phases. The following sections now cover the remaining aspects of this representation in more detail looking firstly at models, then methods and finally phases.

2.3 SOFTWARE DEVELOPMENT MODELS AND PARADIGMS

2.3.1 Early Models

Developing an effective model is a balance between implementing an easy-to-understand life cycle and a process that is flexible enough to address all eventualities [Todd 1993]. Unfortunately this was not achieved by the conventional waterfall approach that is described below.

The software development model defines how the overall software development process is to be performed. It defines the order in which various stages will be tackled through each phase of the development process, and can define some of the methodologies and techniques that will be used within that process. Models define what happens in the phases - not the phases themselves. Broad phases are common to all projects and are covered in more detail in section 2.5.

Unfortunately, what has happened within industry is that time and time again software developments are performed using a specific, established model with which the organisation is familiar. What is advocated is that the model of the software development process should not be preselected until some idea of the product is first achieved.

There are still some organisations that do not knowingly use any explicit development model. Some small software houses interviewed use no apparent model for developments requiring less than one person-month of effort. This seems to be quite common for projects of this size. In these cases the developers can usually be seen to follow a build-and-fix paradigm (see section 2.3.2). Todd (a consultant with 3SL) [Todd 1993] wrote that in his experience of organisations, many software departments were lacking a defined development process, although he provides no specific figures.

Up until the early 1980s there existed only one software development model - the classical waterfall approach. Since then a plethora of other models have evolved which are based perhaps more on the introduction of new methodological ideas than anything else. What has only been tentatively addressed so far is the possibility of combining development models into a suite that allows a developer to choose which model's attributes would suit best a particular project at a particular time. Metamodels represent the next stage in the development of ever more flexible approaches to the development of software systems. Several paradigms currently used within industry are now presented, some of which are incorporated into the metamodel that is described in more detail later.

2.3.2 Build-and-Fix

The build-and-fix, or code-and-fix, model is the earliest approach used to develop software systems. It does not represent any real, explicitly defined model at all, but is used to identify the approach to software development used by many programmers and 'hackers'. According to Schach (1993) this is probably the worst model that can be adopted for the software development process. In this model there are no formal specification stages or requirements analysis. It represents a 'thrown together' coding solution that is subsequently reworked and fixed as required on a repetitive basis until an adequate solution is reached. It is perhaps more widely recognised as the part-time computer hacker's solution to programming software and the model used (although unknowingly) for many other small, one person projects. Three difficulties with this model were identified by Turner (1993):

(i) After several fixes the software becomes difficult to maintain as it becomes poorly structured.
(ii) It often does not match the user's requirements.
(iii) It can be costly to maintain because of its poor structure and lack of definable output that can be tested.

Because of the problems encountered with this unstructured approach several, more detailed models were devised. The earliest of these models was the stage-wise model from which the classical waterfall model developed.

2.3.3 The Stage-Wise Model

The classical software development models are based on a specific phase structure through which a software process cascades to completion. These models were based on the more familiar work of engineering projects and can be traced back as far as the work of Benington in 1956 [Benington 1956]. Benington's model was known as the stage-wise model and, depending on which of the many hundreds of articles read on the subject, there can be anywhere between two and up to or over fifteen specific phases in this model. One respectable view of the classical stage-wise model contains six specific phases (figure 2.3). Detailed explanations of each phase are clearly beyond the scope of this chapter - some such phases having entire books devoted to them. Briefly however:

(i) Requirements Analysis
The idea for the system is identified either by an individual or as a specific requirement that needs addressing within an organisation.

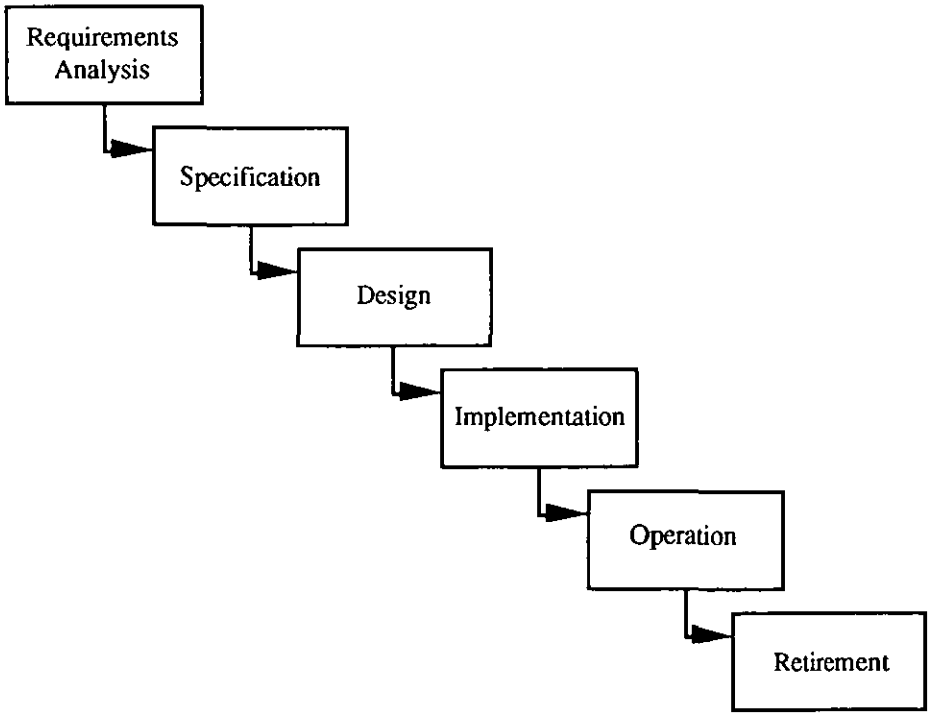


Figure 2.3 A Stage-Wise Model

(ii) Specification phase
Produces a detailed report on what the product should do, not how it should do it. This specification includes any inputs and outputs that are to be produced and any constraints imposed upon the system.

(iii) Design phase
This is where the 'how to do it' document is produced. From the product specification the design phase is used to draw up a detailed design of how the product will go about performing its required objectives.

(iv) Implementation
Conversion of the design, based on the specification, drawn from the initial requirements, into the working product.

(v) Operation
Use of the system, including any enhancements and maintenance work that is required, in its target environment.

(vi) Retirement
The phase out of the product when it reaches the end of its natural life.

One problem with the stage-wise model is that it represents a unidirectional development. In other words, once a stage has been completed the results of that stage become a fixed baseline from which the following stages develop. Usually, problems within a particular development stage are not identified until later in the development cycle. In the stage-wise model any problems within earlier, fixed stages cannot be changed as feedback to them is not identified. One way around this problem is to allow feedback from subsequent stages to earlier ones in the life cycle. The classical waterfall model provides a limited form of this kind of feedback.

2.3.4 The Classical Waterfall Model

The classical waterfall model (or classical life cycle model) overcomes some of the problems of the stage-wise model by allowing some form of feedback to take place between linked stages. Unfortunately, all the waterfall model allows is feedback to one previous stage in the cycle. Each stage, in this case, is identified by its outputs, that feed into the following life cycle stage, rather than the activities that are performed within that stage [Turner 1993]. This model identifies some inherent overlapping between connected stages and allows problems, identified with a previous stage, to be corrected before baselining that stage. A problem with this model is that it again emphasises the

baselining of early stages in the development process, when little is really known about a problem. A lot of work, put into producing the deliverables of an early life cycle stage, could later prove to be wasted when more is known about the problem domain. Figure 2.4 represents the classical waterfall model showing how feedback occurs between each stage.

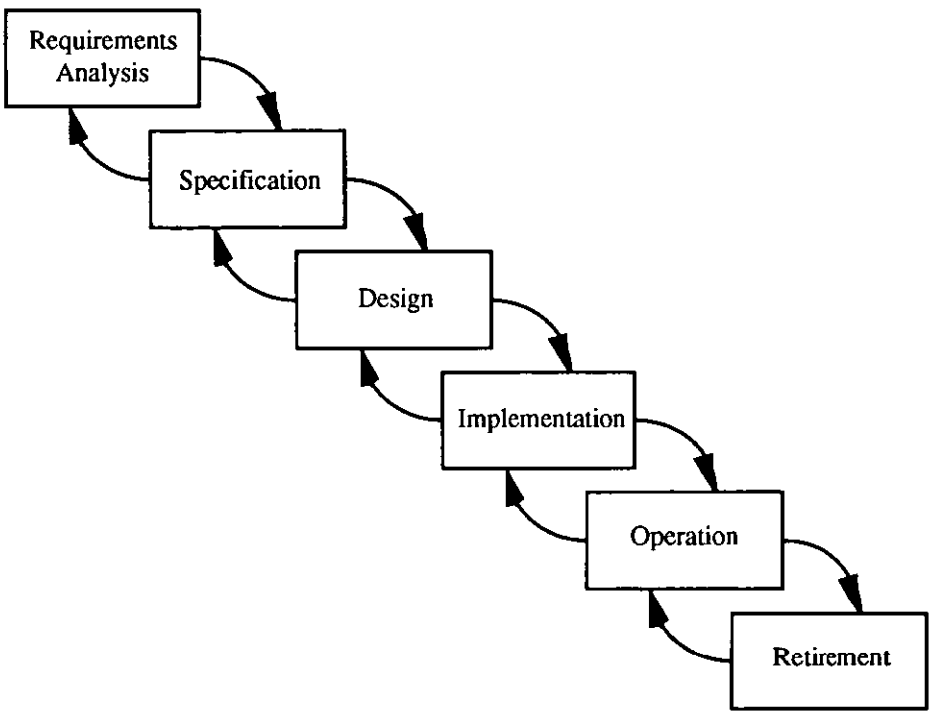


Figure 2.4 A Classical Waterfall Model

It is not by accident that the waterfall model and its derivatives have remained at the forefront of software development. It is due mainly to the inertia of large companies that develop software systems based on these practices [Agresti 1986b]. The aim of this chapter is to highlight the rigidity of these current software development practices, and to propose the use of a more flexible planning technique that allows alternative approaches to be explored by managers.

2.3.5 Prototyping

Prototyping represents a particular technique that, because of its flexibility, has allowed the evolution of whole new set of software development models based on its concepts. Prototyping is a technique whereby information is bought [Macro 1990] at a particular stage of the development process. This purchasing analogy relates to the investment in time and expenditure that provide the developer with the information required. It is when and how the prototype is used that determines which of four prototyping models

is being adopted. Alavi (1984) provided an overview and assessment of the prototyping approach.

(i) General Prototyping
A general prototyping model is the development of a system from an initial prototype developed at either the requirements analysis or specification stage. As the system itself is built up around this original prototype it can be viewed in some ways as a system assembly technique. This technique will be looked at later.

(ii) Throw-away Prototyping
Throw-away prototypes, as the name implies, are discarded once their information has been elicited. Throw-away prototypes are generally used to replace the requirements analysis phase of the waterfall model and are sometimes referred to as rapid prototypes.

(iii) Evolutionary Prototyping
According to Bowen (1990) the evolutionary prototyping model is used to develop a 'production product by the convergence of successive models'. This approach is also sometimes referred to as the *incremental model* [Schach 1993]. In this model the system is delivered to the user in a series of fully operational subsystems. Each subsystem represents a subset of the overall system's requirements and each delivery is a superset of the preceding one. This process continues until the system is completed as a whole. An example of evolutionary prototyping can be found in *Computing* (8 April 1993, p26).

(iv) Incremental Prototyping
Not to be confused with the incremental model, this prototyping approach represents a 'build it twice' ideology [Bowen 1990]. It differs from the throw-away approach in that it is bound by an overall system design and it is not intended as a replacement to an analysis subphase. The incremental prototype is built as closely to the required system as possible and is rewritten each time the system needs to evolve.

2.3.6 Formal Transformations

Sometimes referred to as the *Formal Method Model* this is another model that has grown around a particular development methodology. In this model a functional specification, developed from the requirements specification, is formally converted step by step into a fully operational software system. This conversion can be achieved by formal development languages (for example, some 4GLs developed for this purpose) or by more concentrated coding techniques. Based on formal development methodologies, it is a more exacting software practice and leads to much higher standards of software

safety and software provability. It can remove much of the intensiveness of transforming a specification into a solution through the use of specialised software. The development can also be carried out in a formalised framework that has a precise semantics [Beierle et alia 1986].

2.3.7 Evolutionary Deliveries

This is similar to the incremental prototype in that the final system evolves and grows from a small embryonic core over a period of time. The evolutionary delivery model has the advantage, however, of being able to change its direction as it evolves to reach ever varying customer requirements. This model is sometimes referred to as *exploratory programming* [Sommerville 1993] as the initial releases explore the user's requirements before evolving into the next version of the product. It differs from a build-and-fix model in that there is an initial specification from which to work and it gives a planned sequence of deliverables to provide customer feedback.

2.3.8 Operational Specification

This is similar to a prototype in that it attempts to represent how the final system may look by providing information to the user at an early stage. It does not, like a prototype, need to be developed on the target system that might not exist at that time. According to Agresti (1986a) an operational specification has two advantages. First it separates the development process into problem-oriented and implementation-oriented phases, and second it provides the user with an early executable system model.

2.3.9 Spiral Model

This model, developed by Boehm in 1988 [Boehm 1988], encapsulates some of the better features of the life cycle and prototyping paradigms. It splits the development into four cyclical phases - Planning, Determination (of objectives, alternatives and constraints), Risk Analysis, and Engineering. These phases are performed, in turn, during several iterations of a software evolution. The spiral model works by forcing the risks of a project to be resolved before allowing the next cycle to be pursued. If risks cannot be controlled, or limited to an acceptable level, the project should be terminated there and then. Figure 2.5 provides an outline of this model. In this figure the radial dimension represents incremental costs involved in developing a product, and the angular dimension represents the progress of the project. From high levels of abstraction, each loop repeats previous steps through lower levels of abstraction whilst developing and maintaining the software. According to Sodhi (1991) one advantage of this model is flexibility by providing an ability to encompass any mixture of

specification-oriented, process-oriented or object-oriented approaches. It does not, however, match any existing standards and is still evolving into an acceptable, working model.

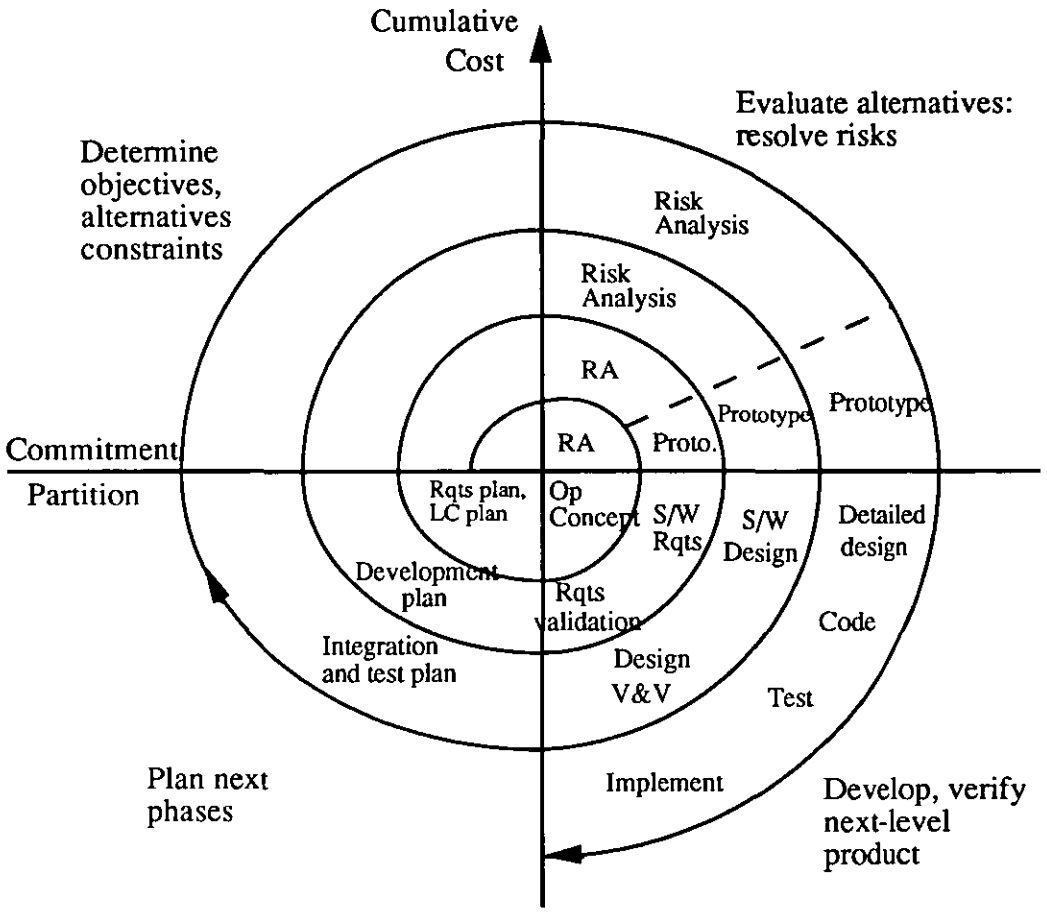


Figure 2.5 Outline of Boehm's Spiral Model

2.3.10 4GT Model

Fourth-Generation Techniques (4GTs) have their roots more as development methodologies and tools, than as particular models. By enabling software to be developed from a much higher specification level, 4GTs allow much faster development of software code. These techniques have therefore led to the establishment of models based on their operation. The 4GT model is a model in which the implementation of the system from the design phase is achieved through a fourth generation technique. Another offshoot of 4GTs is the *automated formal and 4GL model.*

2.3.11 Automated Formal and 4GL Model

This model brings together the more recently developed techniques of 4GLs and formal methods. Figure 2.6, adapted from Pressman (1994), best explains how this model progresses. Following requirements analysis a formal specification can be made (using a formal specification technique). A 4GL can then be used to generate a prototype directly from this specification, that can be optimised and tuned to provide the operational system. This system, like all others, then goes through a period of operation and maintenance before finally being retired.

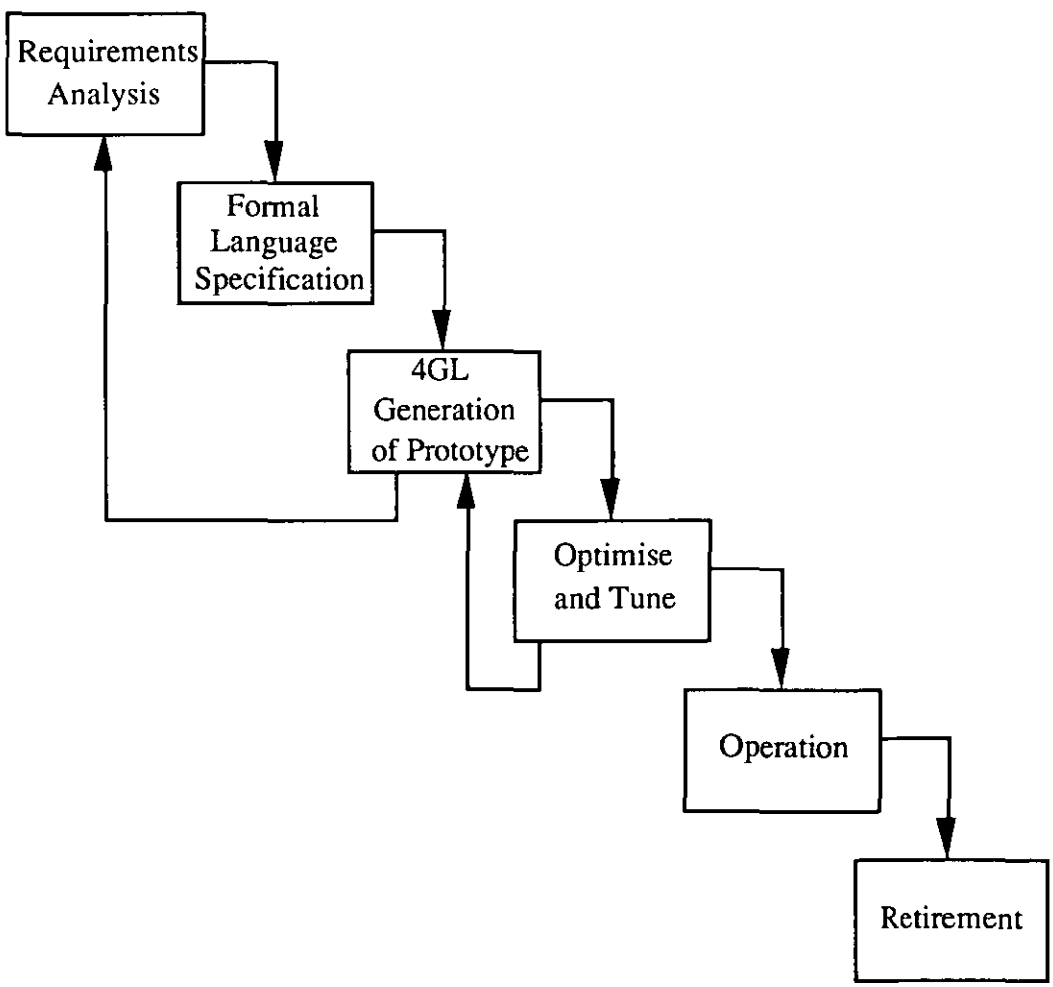


Figure 2.6 Automated Formal and 4GL Model

2.4 SOFTWARE DEVELOPMENT METHODOLOGIES

2.4.1 Existing Methodologies

Although one can draw a distinction between the terms *methods* and *methodologies,* for the purposes of this thesis both terms will be used interchangeably. Strictly speaking, however, one would view a method as being more prescriptive than a methodology as it provides a step by step approach. A methodology, on the other hand, represents a broader approach providing a set of guidelines that can be followed.

Although Todd (1993) announced that very few organisations used any development model, a survey by Spikes Cavell in 1992 [Spikes Cavell 1993] showed that 73% of organisations used some form of software development methodology. A methodology represents a defined way of performing at least one phase of the software development process. It is basically a philosophy that describes the business process. Some methodologies, for example SSADM, are particularly comprehensive and cover several phases of the development process. Others are rather restrictive, having specific rules and protocols that must be followed. Methodologies are also directed towards different aspects of the development process. For example, Prince is a project management methodology, whereas SSADM is a methodology aimed at the design process

If an organisation decides to implement a methodology, as part of its development process, it can adopt it from one of three sources:

(i) Public Domain Methodology
Although the up front costs of a public domain methodology can appear quite low, their long term implementation costs can prove to be rather high. They tend to be quite rigid in their guidelines and can require much expenditure to integrate them into the organisation's structure. Because public domain methodologies are widespread, however, there are many consultancies and much support and training available for them. Typical examples of public domain methods include SSADM, Merise (the French equivalent), and Prince (Projects in a Controlled Environment) by CCTA which is based on Prompt (a government standard introduced in 1983). There is also a Europe-wide method called Euromethod under development that hopes to provide a common methodological standard throughout the EEC [Spikes Cavell 1993].

(ii) Propriety Methodology
A propriety method is supplied and implemented by a single supplier. The user is provided with all the training, consultancy and source material from that one source. The disadvantages with this approach are that it is expensive and the user becomes

reliant on support from a single supplier. Examples of such methods include LSDM by LBMS (Learmonth and Burchett Management Systems), Method/1 by Anderson Consulting, Navigator by Ernst and Young, and Prism by Hoskyns [Spikes Cavell 1993, Falla 1991].

(iii) Home Tailored Methodology
Dissatisfaction with these sources has lead many companies to the development of their own in-house methodologies. These can prove extremely costly to develop but, as they grow out of the companies existing structure, they fit into the organisation's current working practices extremely well. Unfortunately, these in-house methodologies can mushroom out of control without specific guidelines and external support is virtually nonexistent [Falla 1991].

2.4.2 Techniques

There is some overlap between what could be termed a development technique and a development methodology. Generally speaking, however, techniques are not as comprehensive as methodologies, and cover only part of a development phase. A technique can be defined and used within a development methodology but this does not represent a complete solution to an entire phase. Techniques can therefore perhaps, be viewed as sub-methods. They are commonly supported by various tools.

Some examples of techniques and methodologies widely used within industry include PERT (Program Evaluation and Review Technique) and COCOMO [Boehm 1981] that may be used within a planning stage of a project. Data Flow Diagrams, Petri Nets, Z, and Finite State Machines are techniques that assist the specification of a system. Data Flow Analysis, Jackson System Design, Object Oriented Design and Structured System Analysis can be used to design a product. C/SCSC and SSADM are techniques and methodologies discussed in later chapters.

One technique used within the implementation stage is a *system assembly*. As the name implies this is the development of a system by 'bolting' together smaller subsystems. These subsystems could be pre-written, evolved from an initial prototype (the general prototype model), or developed from scratch. Other techniques, that have already been mentioned, are Fourth Generation Techniques that allow software to be developed in more natural languages. Some of these techniques have evolved into specific models that were detailed earlier.

2.5 SOFTWARE DEVELOPMENT PHASES

2.5.1 Overview

Phases are a separate entity to the models that were defined earlier. Whereas models provide an indication of the stages within a process, the phases provide a more integrative view of the project overall. Thus the link between phases and models is that models constitute stages subsumed within phases that are based at the integrative level.

Countless authors have proposed different phase structures in an attempt to represent the flow of software development processes. As mentioned earlier, these structures can consist of any number of phases ranging from two [Grady and Caswell 1986] up to and over fifteen [Bowen 1990]. However, imposing a specific phase structure on a development process can restrict the natural progression of that process. Turner (1993) identified three problems with real life projects that emphasise the importance of not rigidly imposing the life cycle:

(i) Exploratory work on subsequent stages can be required before the current stage is completed.
(ii) Problems encountered in later stages may require reworking of earlier stages.
(iii) Users' requirements are dynamic and may change during the development of the system.

It is not proposed that project phase plans are discarded because a basic underlying structure is still necessary for the effective management of any process.

To show how a phase model can be constructed, an example is provided based on a broad, four stage development structure consisting of analysis, synthesis, operation and retirement (figure 2.7). By using a Work Breakdown Structure on these phases one can subdivide them into smaller stages that relate more directly to a development undertaken. The use of a work breakdown structure to break a project down through various organisational levels (integrative, strategic and tactical) is covered in chapter four.

Figure 2.7 shows some overlap between each of the four phases. This overlap is common within most of the models developed since the early 1980s, since it represents some form of interaction/feedback between the phases. 'In reality there are no clearly defined breakpoints between the stages' [Turner 1993]. This interaction between phases is inherent within the development process and should not be discouraged as it was with the early stage-wise approach.

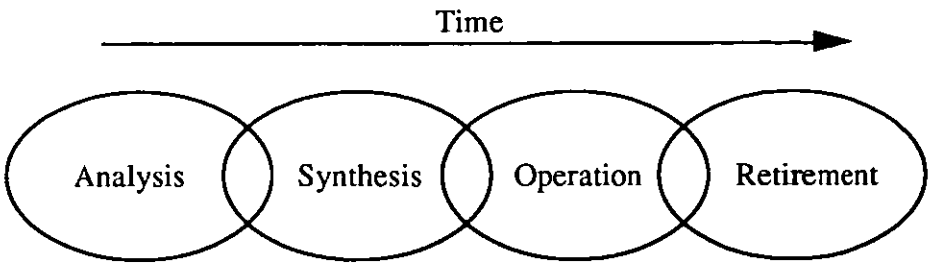


Figure 2.7 A Broad Phase Set

Overlap between the four phases can also be attributed in part to the 'grey areas' within the software development process. For example, there could, perhaps, be some debate as to whether the specification stage should lie within the analysis or synthesis phase. As this is dependent on which development paradigm is employed, and how the information from this stage is used, it is best to show it within the boundaries of both phases. A prototype used as a specification could well be viewed as providing analytical information, whereas a formal specification would not. As it is important not to restrict the process by a phase structure, overlap between the phases defined here is inevitable.

It is interesting to draw a comparison with the phase set defined here, and one defined for use in project management for which software systems represent a subset. Another phase set is that defined by Turner (1993). In Turner's representation of a project life cycle, four phases - Germination, Growth, Maturity and Death - were identified. In this representation the phases relate directly to the project management life cycle, referring to the initiation and subsequent completion of the project itself. These four phases actually refer to the Analysis and Synthesis phases identified in this chapter (Death covering product use). The representation provided in this chapter takes the completion stage one step further and identifies Operation and Retirement as two subsequent steps in the development of a software system. Compared with engineered products, software systems require far more maintenance, constant repairs and upgrades during their operation.

2.5.2 The Analysis Phase

Within this phase one can identify the initial conceptualisation of a project. The project idea can be formed by an individual within an organisation or it can be identified by some form of requirements analysis. One can also identify an amount of project planning where decisions are made as to how a project will be tackled, what resources are used and in what order activities are undertaken (this is covered in more detail in

chapter four). Objectives are also set and milestones are identified. The analysis phase basically identifies the need for a particular product and puts in place a mechanism for developing that product.

2.5.3 The Synthesis Phase

This represents the actual development of a system from the basic concepts. Within this phase managers are responsible for tracking a project as it progresses, and adjusting future expectations accordingly. Often, in many projects, there is a distinct split between the analysis and the synthesis phases although ideally there should be some interaction between them. As a project progresses, more information is uncovered about the problem which can be fed back into the analysis phase. Without any such interaction the benefits of this gain in information would be lost. Control is an identifiable management activity within this phase. This activity is covered in more detail in chapter four.

2.5.4 The Operation Phase

This represents the operation of a final product in its target environment. It includes maintenance and enhancement of a system, perhaps even feeding back information into the analysis or synthesis phases. It can also include adapting software for use on other systems, providing a help desk facility, and marketing the product. It is fair to say that, on the whole, this is the phase to which the least planning and thought is applied even though figures by Macro and Buxton (1987) show that maintenance within this phase can take up to 60% of total project time. It is important that developers recognise the importance of this phase and have some idea of how it will be managed

2.5.5 The Retirement Phase

This represents the phasing out of a system as it reaches the end of its natural life. There can be some overlap between this phase and the operations phase as parts of the system may be phased out, whilst other parts continue to operate.

2.5.6 Further Subdivision of Phases

If this structure is further subdivided it can constrict the development process to which it relates. In this relatively coarse form, it remains an applicable management view of any software development process. Applying a work breakdown structure from this level onwards would depend on the model employed. For example, a formal transformation

model would not include feedback between the analysis and synthesis phases. In contrast, feedback between these phases would be inherent within a prototyping model.

2.5.7 The Rigidity of a Phased Approach

The phases identified above are sufficiently broad to allow software systems to grow within their structure, using any of the development models identified earlier. Generally speaking however, phase structures tend to be so detailed that they constrict the natural progression of a software development process.

Phase structures imposed by development paradigms tend to be adopted in one of two ways. First they can be taken at face value, retaining each phase distinctly and restricting feedback between the phases. Pressman (1994) identifies three implicit problems with this approach. First, real projects rarely follow this sequential flow. Second, it is difficult for customers to state all their requirements at the initial stages, and third, the customer must be patient as they see nothing of the product until its final release.

Technology is now at a level where it is possible for the analysis and synthesis phases to be closely intertwined. This is particularly desirable as it leads to a greater understanding of a problem and the development of a solution that evolves rather than being built from a fixed, predetermined plan. This can stem from a preliminary solution that is produced early in the development and provides a greater knowledge of the problem as a whole. Indeed, it was stated by McCracken (1981) that to prepare a detailed specification, some idea of the solution is first required. This implies greater interplay between the analysis and synthesis phases and in some respects is related to whether one sees the specification falling within the boundaries of either phase. The fixed phase approach must therefore become more flexible to cope with interaction between the phases to accommodate this need.

An alternative approach is for the phase model to be adopted with feedback from sequential, or more separate phases, to earlier phases in the cycle. This can lead to particularly complex feedback models with up to twenty five interlinked phases [Macro and Buxton 1987]. Trying to relate these complex models to individual software developments can be extremely difficult, and, in doing so, establishes it for that particular project.

Unfortunately this approach shows perhaps too much belief in the rigidity of the environment, and also in the precise configuration of the feedback loops themselves. It is important that feedback systems are responsive not only to changes in the system

requirements, but also to proposed changes in the development process itself. The development process should be responsive to any change - be it within the plan, the process, or the solution.

To overcome these problems a more flexible management model is required that does not impose a rigid direction on the process. One solution is to use metamodels that represent a combination of alternative paradigms.

2.6 METAMODELS (OR COMBINED PARADIGMS)

2.6.1 An Early Combined Paradigm

Several different paradigms were detailed earlier that are used in the development of software systems. Feedback from several software houses identified that a specific paradigm is used by these organisations for each individual software development. There are, however, advantages to be had by combining different paradigms within a particular metamodel. The prototyping example from Bowen (1990) goes some way towards emphasising this point. Bowen's prototyping model highlights the advantages of combining two previously separate ideas (incremental and throw-away prototyping) within one particular life cycle model. He showed how the advantages of each idea can be reaped at different stages of the development process.

In 1994 Pressman [Pressman 1994] identified a simple model that combined alternative paradigms within its structure. Pressman identified four paradigms - The Life Cycle, Spiral Model, a Prototyping Model, and a Fourth Generation Model and showed, using a non-deterministic structure, how they could be combined within one model (referred to as a metamodel in this thesis). Figure 2.8, taken from Pressman (1994), is an example of the early idea of combined paradigms. In this chapter, this idea is taken one stage further and it is shown how the benefits of several models can be derived from within one hybrid model called a metamodel.

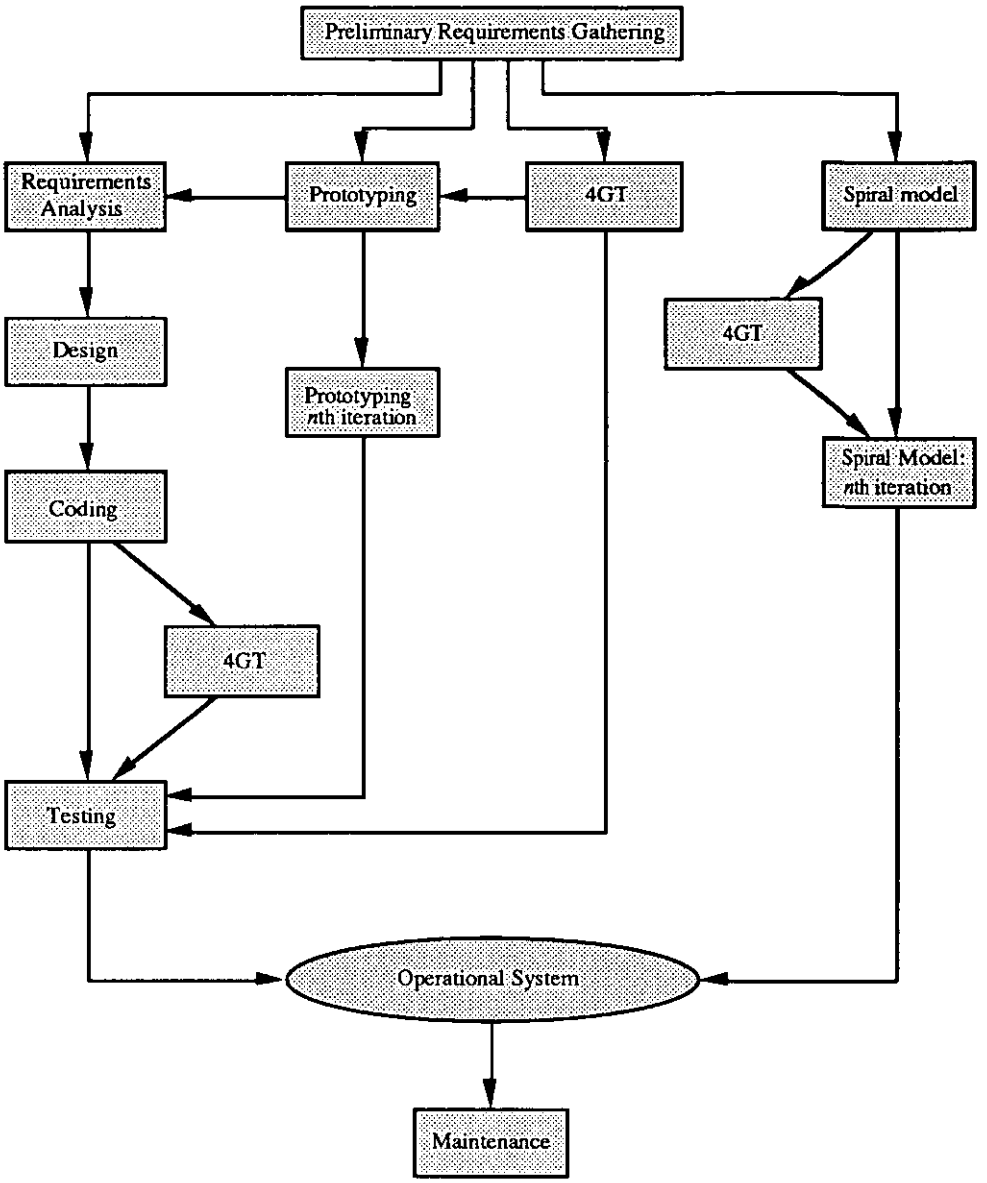


Figure 2.8 Pressman's Early Combined Paradigms

2.6.2 New Metamodels

The concepts behind metamodels are directed at the project manager rather than the systems analyst since these models provide a managerial approach to the development process. Figure 2.9 is an example of the new concept of metamodels. It shows how several software paradigms can be combined within a non-deterministic structure. In this metamodel, eight paradigms have been combined within the structure to represent a strategic level plan. The cells represent subphases (or stages) of the development process, sometimes identified as specific techniques used for their accomplishment. For

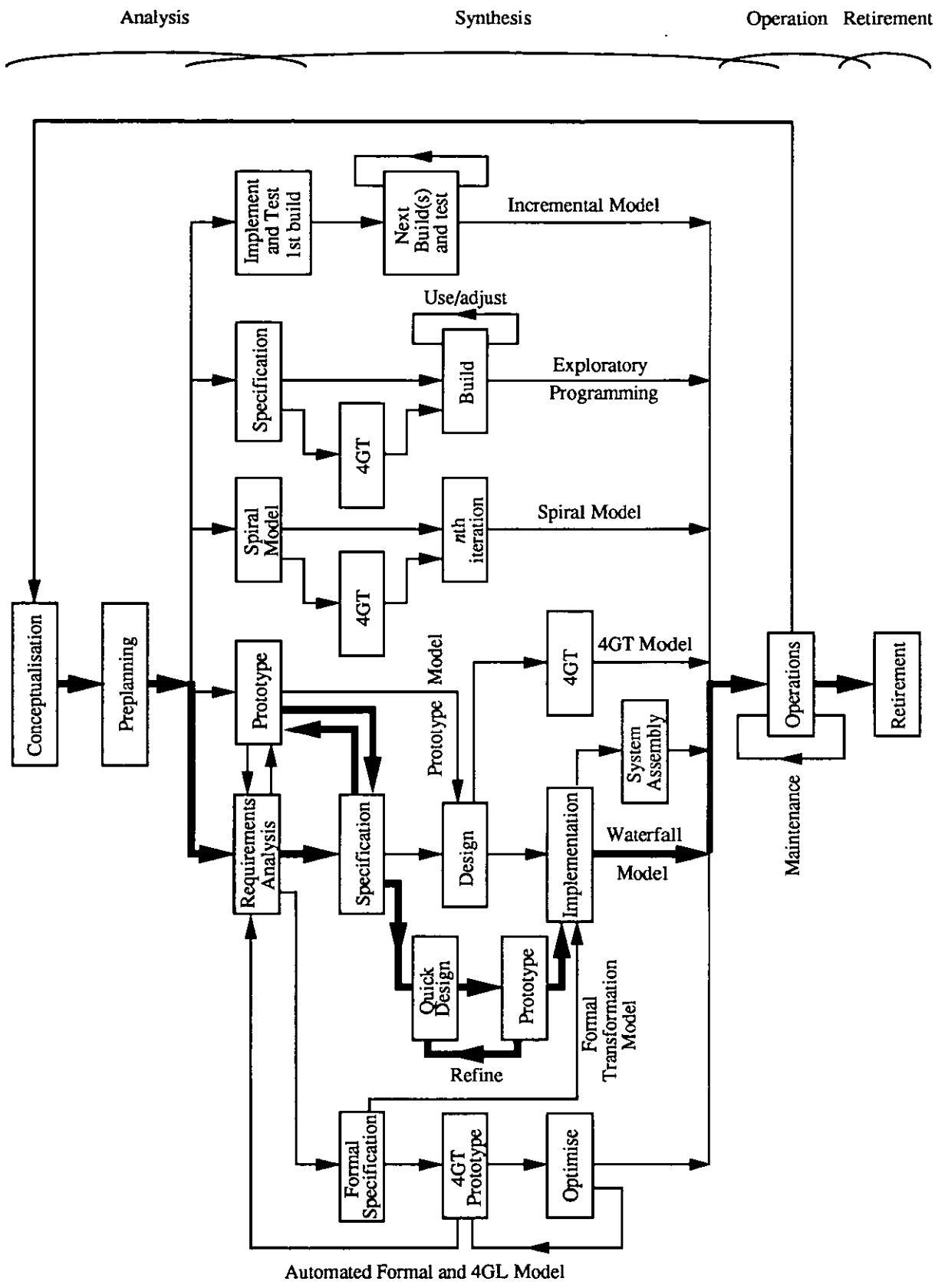


Figure 2.9 A Metamodel

example, the system assembly stage is a technique that is used for the implementation of a system. The stages identified by 4GT represent the use of a fourth generation technique for that particular stage of the development process. The 4GT stages, inherent within certain paradigms (for example the 4GT model identified earlier), can now be accessed from other models within the metamodel. For example, the common waterfall model can access a 4GT stage as part of its implementation without needing to completely replan the project or restructure the development process.

The four phases identified earlier have also been superimposed on this metamodel. They are provided as guidelines to show where overlap between phases is inherent within the metamodel. The analysis phase covers both conceptualisation and preplanning that are inherent within all models. It can also be seen to cover the requirements analysis stage (that can be replaced by some form of prototyping technique) and in some cases the specification stage (especially if this is drawn up as part of the exploratory programming model). The synthesis phase imposed on the metamodel covers the building stages of the system. Again there is some overlap between this phase and the analysis and operational phases. This cannot be avoided as information is often passed between these phases. The operational phase covers modification and maintenance of the system even within the target environment. This undoubtedly overlaps with the retirement phase as parts of a system can be phased out over a period of time.

2.6.3 Advantages of Using Metamodels

Project managers are undoubtedly unwilling to pursue a development without some conceptualised framework in which to work. By incorporating several established models within one metamodel, the advantages of each model can be achieved whilst still providing a project manager with some form of structural framework in which to operate. This combination results in a synergy effect that indicates the sum of the whole is greater than the sum of the individual parts.

All projects begin by some initial idea. The Meliorist Model (figure 2.10) [Lucey 1987] shows how a set of actions, that constitute a project, take an organisation from an existing situation (possibly unacceptable) to a desired situation. The idea to perform a project can come as either a *push* from the existing situation (for example, were a current system is inadequate and has become outdated) or as a *pull* to a desired situation (for example, a new system that would provide benefits to an organisation). The conceptualisation stage, identified within the metamodel, represents this push or pull from an existing situation.

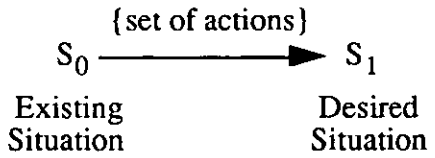


Figure 2.10 The Meliorist Model

After the project has been conceptualised, preplanning identifies the initial direction of the process. For example, it may be decided to build a prototype to complete the requirements analysis of the project. This can be used to develop the specification that in turn can be used to either design the full system or another intermediate prototype. By allowing alternatives to be represented within the metamodel, it enables managers to direct the project according to the current project situation, and not in a direction predetermined at a stage in the life cycle when far less is known about the problem. Because of the non-deterministic branching within the structure of the metamodel managers do not have to decide on the complete development process direction at the outset. The metamodel, in fact, identifies points in the project where decisions need to made, based on the project situation at that time. Managers are, in effect, making decisions as to when to make decisions within the project life cycle. This makes far more sense than attempting to decide initially what will be done later in a project when the project environment may be completely different than it was at the start.

2.6.4 An Example of Metamodel Benefits

From personal contact with the manager of a one person-year project recently, the advantages of a new metamodel approach have been identified. The project, to develop a computer training package for eight year olds, began by using the classical waterfall approach. Two months into the project, as the specification was being drawn up, this approach had to be abandoned as the project fell drastically behind schedule. It was realised at this stage that a prototype was required to determine more clearly the user requirements of the system which were of a highly interactive nature. A general prototyping model was then pursued to the successful completion of the project. Had the project manager not been restricted to a single, predetermined waterfall approach from the start, this 'crashing' of the project after two months could have been avoided. The alternative prototyping paradigm could have been introduced as and when it was required within the metamodel that described the process. This approach would have been a visible, viable option from the outset and, as such, could have easily been adopted into the development process when it became clear that it was required. Figure 2.9 highlights the course this particular project would have taken through the metamodel by

the bolder, arrowed lines. The metamodel thus clarifies all possible options available to a project manager and provides a more visual representation of a project's progress. Chapter seven provides more detail of this example.

2.6.5 Tracking the Development Process with Metamodels

One other benefit of the metamodel is its ability to provide project tracking even when a software development has made several diverse changes. In the example studied (the computer training package for schools) the project baseline plan became so disjointed from the secondary plan (which included prototyping) that it became particularly difficult to monitor its progress against this plan. The baseline plan no longer represented the project direction and did not reflect the project's stages any more. A secondary baseline plan was used to monitor the project's progression to its successful completion. Many would argue that a project should be measured against an initial baseline. In software developments, it is possible that the current project direction has digressed so much from the baseline that a direct comparison is difficult to provide. Because the metamodel provides all possible project directions at the outset, the software development will follow one of these routes. The project can therefore be related to this initial baseline plan as it will not digress outside the metamodel. Baselines provided by the waterfall model, for example, do not allow this. The baselines in these cases are fixed at each previous life cycle stage and not as a flexible baseline at the start.

2.6.6 The Way Forward with Metamodels

The metamodel defined in this chapter is not a definitive model and represents one of many such metamodels. Organisations currently using perhaps one or two different development approaches separately should be able to combine these within their own metamodels, reaping the benefits that all their models can provide. The way forward is to develop metamodels that provide the required benefits to an organisation to which they are suited.

What is also required is a means whereby this type of model can be managed and controlled. With reference to figure 2.2, a support element is required that assists a management activity (planning) for any particular metamodel. The management technique introduced below provides this support.

2.7 A FLEXIBLE PLANNING TECHNIQUE

2.7.1 Generalised Activity Networks

Perhaps the most common approach to planning projects, and consequently software developments, is for a manager to use some form of project management software tool. These tools are based on 1960s' ideas (for example, PERT) enhanced with 1990s' technology (for example, Windows™). Generally speaking the more one pays for these packages the prettier the screen looks, the more powerful the input and output capabilities are, the more detailed are the results, and the larger the project(s) one can manage. Their underlying concept remains the fixed project development structure that, as far as software development is concerned, now belongs in history. The alternative to these fixed planning techniques are Generalised Activity Networks that also originated in the late 1960s but which have never really caught on.

Generalised Activity Networks operate in a similar way to standard PERT networks. They represent projects as a series of tasks symbolised as interconnected arrows or nodes (they are defined as both Activity-on-the-Node and Activity-on-the-Arrow networks in the next chapter). Generalised Activity Networks differ from ordinary Probabilistic Activity Networks in their definition of node input and output characteristics. PERT networks insist on a deterministic node logic which implies that all activities must occur, successfully in sequence, for a project to complete. Quite clearly, this deterministic structure would be of little value for planning a software development that uses a metamodel as a framework. It would not be able to cope with alternative routes offered by a metamodel within its representation. Generalised Activity Networks, on the other hand, allow either deterministic or probabilistic branching to be defined. This allows a more flexible project plan to be defined covering various phases determined implicitly by whichever development model is being employed.

2.7.2 Benefits of Generalised Activity Networks

Another interesting facet of Generalised Activity Networks is their ability to handle loops. If loops are formed in ordinary PERT type networks, a situation exists where activities cannot start until after they have completed - an unacceptable logic. The probabilistic nature of a Generalised Activity Network, on the other hand, allows feedback (either probabilistic or defined) from activities to earlier stages in the project plan. This ability provides an ideal representation for situations in metamodels where feedback is inherent.

Generalised Activity Networks provide a means of planning variability and can be nested in the same way as ordinary activity networks and bar charts. At their uppermost level they can be used to plan and control the stages of a metamodel at a strategic organisational level. An Activity-on-the-Arrow representation of the metamodel in figure 2.9 is shown in figure 2.11. In order to keep things simple at this stage standard, deterministic PERT nodes have been used. Obviously the loops and probabilistic branches shown in this structure are not possible in an ordinary PERT technique. Different node logics need to be defined so that they can cope with such problems. These definitions are provided in chapter three. In constructing this network another apparent link came to light between the prototype and the quick design stages. This link, although not identified in the initial metamodel, should be an option as a prototype-based specification could clearly lead to a quick design.

The activity network of figure 2.11 not only provides a project manager with a visual representation of a project plan based on the metamodel but its analysis also determines likely costs, durations and risks involved in a project overall.

Not only can Generalised Activity Networks be used for planning at the upper levels of the development process (at the metamodel level) but they can also be used to plan at the lower tactical levels (nesting). At the lower levels of the development process it is possible to use Generalised Activity Networks to plan how particular stages of a metamodel are performed. How the implementation stage of a software development can be planned with a Generalised Activity Network is provided as an example in the following chapter.

2.8 CHAPTER SUMMARY

2.8.1 Conclusion

The software crisis has been around now for well over twenty years due, in part, to organisations using development models that originated from completely different project fields. It is now recognised that software is much more evolutionary than engineered products and it is therefore necessary to adapt development processes accordingly. What has been introduced within this chapter is the idea of hybrid paradigms, or metamodels, that allow software to grow within their structure. They leave managers with the project visibility they require, and allow more applicable development routes to be pursued as and when required. By combining several different paradigms within one structure, the benefits of each can be gained within one development (providing that a project is allowed to evolve along its most logical route).

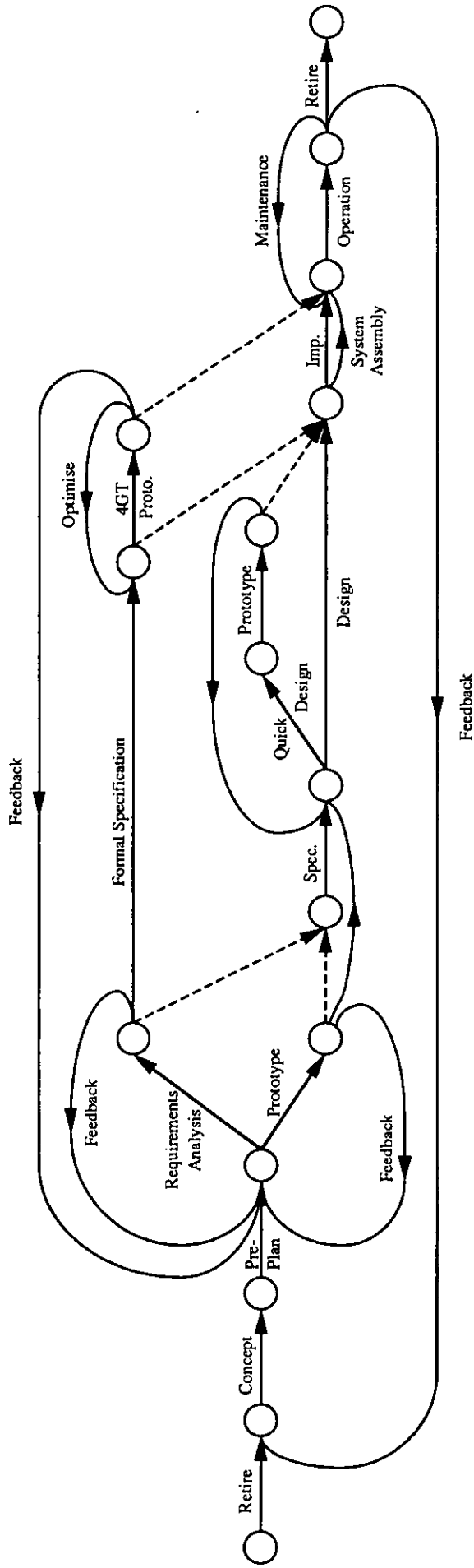


Figure 2.11 Generalised Activity Network Representation of the Metamodel

Generalised Activity Networks have also been introduced that help the management activities associated with planning and controlling more flexible software systems development. It is perhaps too ambitious to assume that developers will take on board a more flexible approach to managing projects without an established means of support. It is anticipated that using more flexible management planning techniques, like Generalised Activity Networks, may have to predate the employment of more flexible metamodels. Managers may be unwilling to take such bold steps without the support of a system with which they are comfortable.

To imply that managers are to give a free reign to the development of software and let it evolve of its own accord would be wrong. What is suggested in this chapter is that a more flexible approach is needed for managing software development projects with the support of a more realistic planning technique. Software developments still require some form of project management and planning, albeit a flexible one.

2.8.2 Future Work

Two areas still require developing from this initial research:

- Development of software dependent metamodels
- Development of organisation specific metamodels

The first of these areas identifies that different application domains have different development needs. Metamodels for specific problem domains therefore need developing from the broader metamodel presented in this chapter. Organisations have also employed particular methodologies for their own domain specific developments. Metamodels represent models that can be dedicated to the needs of specific organisations. Again, these metamodels require some further development.

A discussion of other possible research areas related to this work is covered in chapter seven.

CHAPTER 3

Generalised Activity Networks for Project Management

CHAPTER PREFACE

In this chapter Generalised Activity-on-the-Arrow (presented in Dawson and Dawson (1994a)) and Generalised Activity-on-the-Node Networks are defined. The chapter begins by studying earlier developments in this field before providing a conclusive definition of nodes in Generalised Activity-on-the-Arrow Networks. This work is then developed to provide a definition of a Generalised Activity-on-the-Node representation that also allows Logical Dependency Constraints to be applied. The Generalised Activity Networks defined can be used as tools to manage both software developments and general engineering projects where uncertainties exist.

CHAPTER KEYWORDS

Generalised Activity Networks, Activity-on-the-Arrow, Activity-on-the-Node, Logical Dependency Constraints

3.1 INTRODUCTION

3.1.1 Scope

'The price paid for a more realistic representation is that the model is generally far more difficult to analyse' [Moore and Clayton 1976].

Chapter one provided a general introduction to the different kinds of activity network that are available to assist a project manager. Generalised Activity Networks are the most powerful of the three activity networks available. Although the analysis of Generalised Activity Networks can be somewhat difficult (this is addressed in the following chapter) they do provide a more realistic way of mirroring real life projects.

A study of previous literature in this field showed little consistency between any of the Generalised Activity Networks defined. This is due mainly to the fact that none were adopted by industry and consequently no recognised, industry-wide standards emerged.

Some Generalised Activity Networks also provided rather ambiguous definitions and overly complex representations, to which project managers had difficulty relating. What has been required is a conclusive definition of Generalised Activity Networks. This chapter provides such a definition and prepares the basis for a standard.

All previous literature devoted to the subject of Generalised Activity Networks (except P-GERT [Pritsker 1974]) is concerned with an Activity-on-the-Arrow representation. Later in this chapter, current thinking in the field of project management is addressed and a Generalised Activity-on-the-Node representation is derived.

This chapter begins by clarifying all possible input and output forms of Split Node Logic nodes in Generalised Activity-on-the-Arrow Networks. The chapter goes on to define the three basic Unit Logic Nodes that are required and details some other Generalised Activity-on-the-Arrow Network functions. This logic is then developed into a Generalised Activity-on-the-Node representation that reflects the direction in which most project management tools today are heading. By finally encompassing all possible node input/output forms in a Generalised Activity Network presentation, a representation is provided that can form the basis of an industry-wide standard.

Because of the flexibility provided by Generalised Activity Networks they offer several advantages over standard Probabilistic Activity Networks. These advantages will become apparent when specific examples (of Generalised Activity Networks being used) will be studied later.

3.1.2 History

The first recognised Generalised Activity Network was introduced by Eisner in 1962 [Eisner 1962] when he developed a 'Decision Box Planning and Scheduling' technique for research projects. This involved simple dichotomous choices at each node in a network. Another early networking technique with some form of in-built probabilistic option was *Decision CPM* [Crowston and Thompson 1967]. This technique explicitly identified alternative ways of performing different tasks (with different costs and durations) in a network diagram. More generalised developments were made by Elmaghraby in 1964 and 1966 [Elmaghraby 1964 and 1966]. Elmaghraby viewed nodes from the perspective of both input and output logics (called Split Node Logic or SNL). Unfortunately, Elmaghraby was somewhat unclear in his definition of node output logic. In most cases he assumed that only one activity can be performed from any node (termed Exclusive-Or in this thesis), but in certain cases more than one activity was performed. In figure 3.1, taken from Elmaghraby (1966), node 2 can only be realised if both activities *a* and *b* are successfully completed. This is in contradiction to

Exclusive-Or logic that was assumed later in his paper when he analysed some example networks.

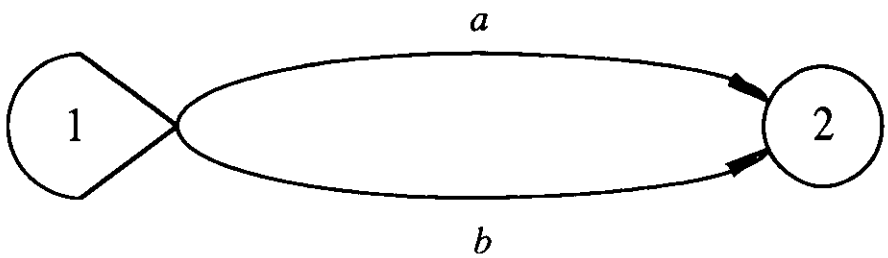


Figure 3.1 Elmaghraby's Example Network

1966 also saw the introduction of the first of a series of Generalised Activity Networks called GERT (Graphical Evaluation and Review Technique) [Drezner and Pritsker 1966, Pritsker and Happ 1966, Pritsker and Whitehouse 1966, Whitehouse and Pritsker 1969, Pritsker 1979, Pritsker and Sigal 1983]. GERT was based on the ideas introduced by Elmaghraby [Elmaghraby 1964 and 1966] and consequently inherited the same deterministic and Exclusive-Or node output characteristics. GERT metamorphosised into several domain-specific types. One of these was a purely Exclusive-Or input and output form, called GERTE, that enabled analysis to be performed by flowgraph theory (this form of analysis is presented in Appendix C). It was clear, in this Generalised Activity Network, that the technique was being restricted by analysis limitations. A simulation version of GERT, called GERTS, was therefore developed that included the deterministic and Or-type inputs and outputs defined by Elmaghraby (1964). GERTS progressed to incorporate resource requirements (without scheduling) and an Activity-on-the-Node representation called P-GERT [Pritsker 1974]. P-GERT is the only known reference to a Generalised Activity-on-the-Node representation.

According to Moore and Clayton (1976) the features available in GERT that do not appear in PERT (ie standard Probabilistic Activity Networks) are:

- Probabilistic branching
- Network looping
- Network modification during execution
- Multiple sink nodes
- Multiple node realisations
- Specified activity releases

- Multiple probability distributions
- Multiple types of node input

Each of these features is available with both the Generalised Activity-on-the-Arrow Network and the Generalised Activity-on-the-Node Network presented in this chapter.

The only development of Generalised Activity Networks since the time of GERT has been the introduction of a particularly powerful technique called VERT (Venture Evaluation and Review Technique) [Moeller 1972, Moeller and Digman 1981, Kidd 1991]. VERT 'can offer a flexible tool for the strategic analysis of a project' [Kidd 1990]. The VERT technique continued with the split node logic of earlier techniques but also introduced a *unit logic node* in which the output arcs from a node are linked in various logical ways to the input arcs. Unfortunately, VERT has three shortcomings. First, it again does not fully encompass all the possible probabilistic input and output forms that can be defined for activity network nodes. It could not, therefore, mirror the example provided later in this chapter. Second, because of its rather rigid and specific design, it does not appear to be a natural progression from the PERT technique with which most project managers are familiar. To take such a conceptual step is beyond the scope and commitment of most managers. Third, VERT does not incorporate the ability to handle loops - an inherently practical ability of Generalised Activity Networks - that enable repetitive activities to be planned and controlled.

3.1.3 Generalised Project Management Tools

Two project management software tools, used by industry today, that incorporate some form of Generalised Activity Network facility are Risnet™ (1993) and Monte Carlo™ (1993). Available for DOS version 3.2, Risnet™ provides twelve possible activity temporal functions and several node input and output definitions. It is based on the Activity-on-the-Arrow approach but unfortunately, it does not provide an ability to model loops and provides none of the powerful features associated with Unit Logic Nodes (defined in detail later). The second project management tool, with some form of Generalised Activity Network representation, is Monte Carlo™ 2.0 - used in association with Primavera Project Planner® (1994). This provides only very basic probabilistic branching, concentrating more on its ability to model several different activity temporal functions. As Monte Carlo™ is based on the output from Primavera Project Planner™ it represents an Activity-on-the-Node approach. However, its probabilistic branching is limited to only two types - Conditional (Exclusive-Or in this thesis) and Probabilistic (Independent-Or in this thesis).

3.1.4 Use of Generalised Activity Networks

Although a combination of all previous Generalised Activity Network definitions could handle all project scenarios, they never became an established project management technique. This is can be attributed to four reasons.

First, at the time of their development there was no prospect of software technology to support their analysis requirements. With the advent of software technology, and the introduction of personal computers in the 1980s, software planning tools were developed based on the more established network techniques of PERT and CPM. Had the technology to both create and analyse Generalised Activity Networks, been available in the early 1970s, it is probable that they would have developed as the standard planning tool for most projects. A Generalised Activity Network has more appeal to project managers because of its flexibility and its ability to mirror projects more accurately than the PERT technique.

Second, it is difficult to analyse standard Probabilistic Activity Networks without adding the complication of stochastic activity nodes that are found in Generalised Activity Networks. Although a project manager can produce a reasonable plan for his/her project using a Generalised Activity Network, without suitable analysis the plan would be virtually useless. The most productive way of analysing activity networks (both Probabilistic and Generalised Activity Networks) is by simulation techniques. Simulation avoids the computational expense of implementing excessively complex multivariate integration solutions and conditional probabilities. It provides accurate results of project characteristics in a reasonably short time and provides a practical means by which analysis of Generalised Activity Networks could be incorporated into software planning tools. Chapters four and five provide a more detailed discussion of activity network analysis.

Third, it can be difficult to estimate the probability factors on which activity generations are based. This can only improve through experience based on previous project results. If managers were able to make regular use of a software tool for Generalised Activity Network planning, they would become more used to this kind of prediction and the problem would diminish.

Finally, human nature means that it can be difficult to persuade managers to incorporate failure into their plans. The possibility of failure and the maturity to deal with its outcomes must be incorporated into Generalised Activity Network plans. This allows better analysis of risks than would otherwise be possible in standard activity network approaches. Again, this problem will reduce once managers have gained experience of

Generalised Activity Networks.

Another possible reason why Generalised Activity Networks never became popular was noted by Schonberger (1981). He wrote that managers tend not to have adopted a GERT to assist with project planning because they are possibly bewildered by conflicting theories as to which probability distribution to assign to project activities with the technique. It is certainly the case that overly complex systems can be more of a hindrance than a help. The Generalised Activity Network defined within this chapter can be used, if required, by managers in a simple Probabilistic or Deterministic Activity Network form. The more powerful planning facilities that are available can be introduced as and when required.

3.2 GENERALISED ACTIVITY-ON-THE-ARROW NETWORK DEFINITION

3.2.1 Essential Features

Like all Activity-on-the-Arrow techniques each activity, or stage, in a project is represented by an arc (ij) that connects two nodes i, (its preceding node), and j (its succeeding node). Because of this, Activity-on-the-Arrow networks are sometimes referred to as *I/ Networks* [Turner 1993]. Each arc is directed, that is, it shows a flow of information between the two nodes that it links. Each arc has associated with it a time, a cost, a performance measure, and a probability. The time, cost and performance measures usually take the form of distribution functions (as in PERT). The probability, represented by p_{ij}(starts), is that activity ij is performed given that its preceding node, i, is realised.

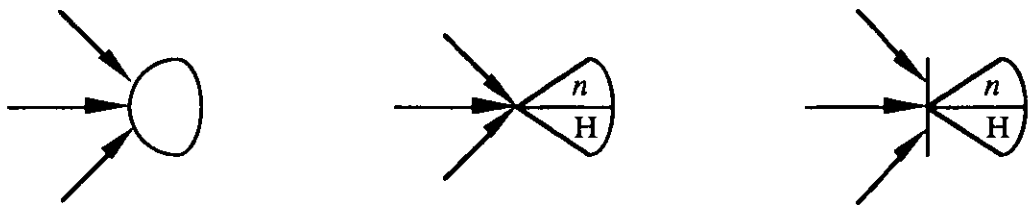
Activities, in this definition of Generalised Activity Networks, can reside in one of three states at any given time. First, they can be in an *active* state that means they are currently being performed. Second, they can be in an *eliminated* state meaning they have being logically eliminated from the project at that time and will not be performed during that project run. A *neutral* state represents the third state in which activities, not in either of the first two states, reside. Although not strictly a state, a *completed marker* can also be applied to an activity to signify its successful completion. This marker is required for the Arc Constrained logic that is detailed later in this chapter.

Before each possible node representation was developed, all possible node realisations were determined and compared with previous literature in the field (for example, Drezner and Pritsker (1966), Moore and Clayton (1976)). Nodes in a network represent events that have any number of incoming and outgoing activities. The main

focus of attention in Generalised Activity Networks is at these nodes. The input and output forms are detailed below with examples based on typical problems encountered when planning software development projects.

3.2.2 Node Input

Each node can receive incoming activities in one of three ways; And, Or and Exclusive-Or (Figure 3.2).



And Or Exclusive-Or

Figure 3.2 Generalised Node Inputs

In the following definitions p(j) represents the probability that a node, j, is realised given that p_{hj}(completes) and p_{ij}(completes) represent the probabilities that incoming activities hj and ij complete successfully. This can, of course, be extended to m incoming activities (m>0).

(i) And

All activities that are incoming to the node must be completed before the node can be realised. This is the PERT form of the node input.

Example: In a software development, testing of modules must complete successfully before the work of integrating them can begin.

$$p(j) = p_{hj}(\text{completes}) \cap p_{ij}(\text{completes}) \qquad (3.1)$$

(ii) Or

This node requires n incoming activities to complete successfully before it can be realised. In many cases n=1. This is taken as the default value and can be omitted from the node. This definition implicitly accepts that a node can be realised every time n incoming activities complete. Equation 3.2 represents a basic case when n = 2. The different codes (represented by H in figure 3.2) applied to the node affect this reasoning as follows:

H = *Blank* - as for the above definition. The node is realised when n incoming activities complete. It can be realised as many times as required and does not rely on outgoing activities completing (as code Q).
Example: For n=1 this could represent a debugging process in a software development. Each time a bug is found in a program a debugging process would be initiated.

$H = C$ - The node is realised when n incoming activities complete successfully. All other activities entering the node are allowed to complete in their own time. The node can be realised once only.
Example: Where a number of system performance tests are being executed, although one test fails, it may be better to complete any remaining tests as they may provide more information and perhaps highlight more serious problems.

$H = H$ - Cancel all remaining, active incoming activities when the node is realised and eliminate all others.
Example: When searching for a particular problem in a system by several methods, as soon as the problem is located all other search methods are terminated.

$H = W$ - The node is realised when all active, incoming activities are completed successfully and all other incoming activities have been eliminated. In other words, at least n activities entering the node must have their completed markers set, and all others must have been eliminated. This differs from the And input which would not allow incoming activities to be eliminated from the project. This node can only be realised once.
Example: When performing user testing, all user tests must have completed (either successfully or unsuccessfully) before beginning work on a performance report.

$H = Q$ - This represents a type of queuing node that represents a point in a project at which only n completing activities can be dealt with at any one time. This is equivalent to the Blank-Or input except that all outgoing activities from the node must have completed (or been eliminated) before the node can be realised again. In this case it ensures that repetitive work does not run in parallel with itself.
Example: When performing user testing, after a user test completes a report is written on that test. Work on the next report cannot begin until the first one is complete even though new results have come in.

$$p(j) = p_{hj}(\text{completes}) + p_{ij}(\text{completes}) - p_{hj}(\text{completes}) \cap p_{ij}(\text{completes}) \qquad (3.2)$$

(iii) XOr

Exclusive-Or. In previous Generalised Activity Network literature, this represented a rather unrealistic input characteristic that would cancel the node if more than one incoming activity completed successfully. It is difficult to envisage uses of an Exclusive-Or node in this originally defined form. This ability can be achieved, if required, by using the more powerful Unit Logic Nodes that will be described in the next section. A far more realistic use of the Exclusive-Or is to constrain the Or node input above to exclusive activity completions. In other words, for the definitions above, n different activities must complete successfully before the Or node logic can be interpreted, and not one activity completing successfully several times (which is possible). This matches the Type A input of the GERT technique but it incorporates the flexibility of the other node input definitions defined above. In its original Exclusive-Or form, p(j) would be calculated from equation 3.3.

Example: Testing of different modules must complete successfully. It is no use having one module passing several tests whilst other modules fail theirs.

$$p(j) = p_{hj}(\text{completes}) + p_{ij}(\text{completes}) - 2[p_{hj}(\text{completes}) \cap p_{ij}(\text{completes})] \quad (3.3)$$

Two possible input criteria that have not been defined are a Cost/Time dependency and a a multiple realisation W-Or. No real examples could be defined for a multiple realisation W-Or. Adding this criteria to the definitions above, however, would only require the definition of another input code (for example, $H = W2$). The Cost/Time dependency possibility is defined as part of a node's output and is examined below.

3.2.3 GERT

In simulation analysis of GERT networks [Drezner and Pritsker 1966, Moore and Clayton 1976], for example the GERTS-IIIZ program maintained by Pritsker and Associates Incorporated, an alternative definition of the node input is achieved by applying different codes to the node (figure 3.3).

α Number of activities required for first realisation of the node.

β Number of activities needed for second and subsequent node realisations. $\beta = \infty$ implies the node will be realised once only.

δ Node identification number.

ω Node realisation code. This takes one of four values:

- Blank — Normal realisation occurs - The node may be realised each time by the same incident activity repeating more than once.
- Type A — Different incoming activities are needed to

activate the node.

Type H Cancel work of remaining incident activities when the node is realised.

Type U A combination of type A and type H.

Examples of these definitions in a software development project are:

$\alpha = 2,\ \beta = 1$ Module integration - two completed modules are required before integration can begin. Subsequently only one more module is needed each time for integration to continue.

ω = Blank A simulation test system (ie with repetitive testing) can use results from the same, or different test routines to test a system.

$\omega = A$ Testing of all modules must be successful - it is no use having one module testing successfully several times if others are to fail.

$\omega = H$ Cancel remaining testing when one unit test fails. After reprogramming, all units will need retesting from scratch and so there is no point in testing them further at this stage.

$\omega = U$ A combination of examples A and H above.

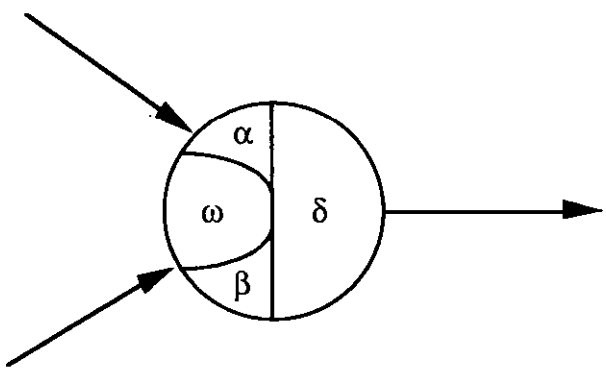


Figure 3.3 A GERTS-IIIZ Node

3.2.4 GERT Time Statistics

As a network is simulated it may be desirable to collect various time statistics associated with the network. To this end it is possible to assign particular time statistic codes to certain nodes. Figure 3.4 shows where this code, τ, is located on a node with the corresponding values explained below. Thus, if a network was simulated, the required time statistics could be acquired from the network.

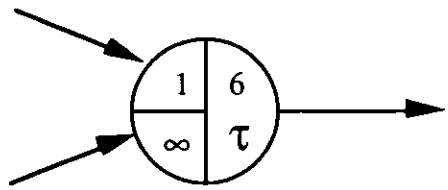


Figure 3.4 Another GERTS-IIIZ Node

$\tau = F$ Time of first realisation of the node (from the source node).

$\tau = A$ Time of all realisations of the node (from the source node).

$\tau = B$ Time between realisations of a specific node.

$\tau = I$ Time interval from a mark node to a statistical node.

$\tau = D$ Time delay from first activity release at a node until the node is realised.

$\tau = M$ Mark node - forces future statistics to be referenced from this node as opposed to the source node.

3.2.5 Node Output

Once a node has been realised the activities emanating from it can begin depending on the form of the output definition. As shown earlier in this chapter, previous literature [Elmaghraby 1964 and 1966] assumed an ambiguous Or-type node output. In this section the output types are clarified by using distinct node representations. The output form of a node can actually take one of five forms: Deterministic, Independent-Or, Exclusive-Or, Dependent-Or, or Cost/Time Dependency. Examples of these node styles are provided in figure 3.5 with probabilities shown as examples.

(i) Deterministic

All activities emanating from this node begin when the node is realised. This is the PERT form of node output.

Example: In a software development, the coding of all modules can begin after the specification is complete.

(ii) Independent-Or

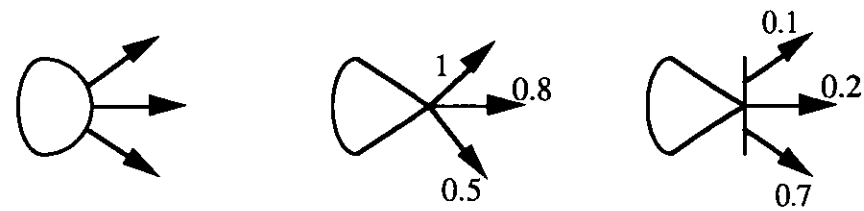
All activities emanating from the node start independently with a given probability. This can lead to a situation where all or, at the other extreme, no activities are started when the node is realised.

Example: Following user trials the next activities may be; alter user manual, alter code or any combination of the two.

(iii) Exclusive-Or

In this case only one activity emanating from the node can be activated. The sum of the probabilities of the activity actuations in this case must equate to one, as the probability of one activity being chosen to start is dependent on not selecting any of the others.

Example: A system test could result in performing a success activity or failure activity but not both.



Deterministic Independent-Or Exclusive-Or

Dependent-Or Cost/Time Dependent

Figure 3.5 Generalised Node Outputs

(iv) Dependent-Or

The activities emanating from this form of node are in some way dependent on one another. As for the independent case, each activity has its own probability of occurring, but the combined probability no longer equates to the product of the individual activity occurrence probabilities (equation 3.4). If this were the case then the node would be replaced by the independent form.

Example: Following user trials the next activities may be; improve the software speed and/or reduce the memory costs. Generally speaking, it may be particularly difficult to attempt to improve both these problems as in some cases they affect one another. The combined probability of even attempting to start both these activities together is probably less than the product of both activities combined.

$p_{hj}(starts) \cap p_{ij}(starts) \neq p_{hj}(starts)*p_{ij}(starts)$ (3.4)

In figure 3.5 (Dependent-Or) one can see there is no explicit representation of the combined probability value. This could be somewhat of a drawback. An alternative way of representing a Dependent-Or output is to use dummy activities and nodes so that the probabilities can be shown explicitly. Figure 3.6 shows how activities A and B, which are dependent in some way on one another, can be represented using an Exclusive-Or output and several dummy activities. The price one pays for this explicit representation is an overly complex replacement to the simple Dependent-Or output node and this would become even more complex if several outgoing activities were dependent on one another.

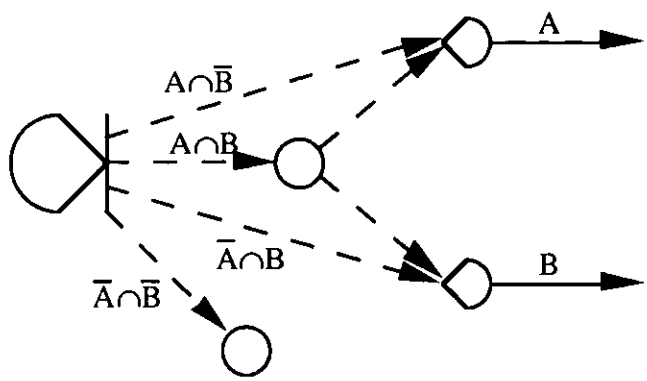


Figure 3.6 Messy Alternative to Dependent-Or Output

(v) Cost/Time Dependency

The output activities from this node are dependent on the times and costs accrued to date. This is particularly useful in planning for situations in which costs and times significantly affect the course of a project. For example, if a project is overrunning its estimated costs by 50% at a particular stage in its development cycle, it may be decided to scrap the entire project. Being able to plan for this terminal possibility beforehand provides the project manager with a clearer means of risk analysis.

Another suggestion for Cost/Time dependency could be to apply these restrictions to the input side of a node. For example, a node could be activated after X days into a project. In this case the activities emanating from the node are dependent on this cost/time dependency and it is better to apply the code to the output side of the node for consistency. Having this logic on the output side of a node also allows it to be used as a source node - for example, activities A and B are started three days after the project begins.

With each node having three possible forms of input and five possible types of output, there are fifteen unique SNL node types. Table 3.1 illustrates how the input and output styles combine to form all the available nodes.

Output \ Input	And	Or	XOr
Deterministic			
Independent Or			
Exclusive Or			
Dependent Or			
Cost/Time			

Table 3.1 All Possible SNL Nodes

3.2.6 Unit Logic Nodes (ULNs)

Unit Logic Nodes (ULNs) were first introduced in the VERT technique by Moeller (1972). The ULNs defined in this work were given particularly complex names such as 'Time Cost Performance Link Escape' and 'Partial Time Cost Performance Link Escape'. ULNs link the output activities from a particular node logically to that node's incoming activities. Only three types of ULN require definition - shown in figure 3.7. The output activities from these nodes are activated as soon as their input criteria are satisfied.

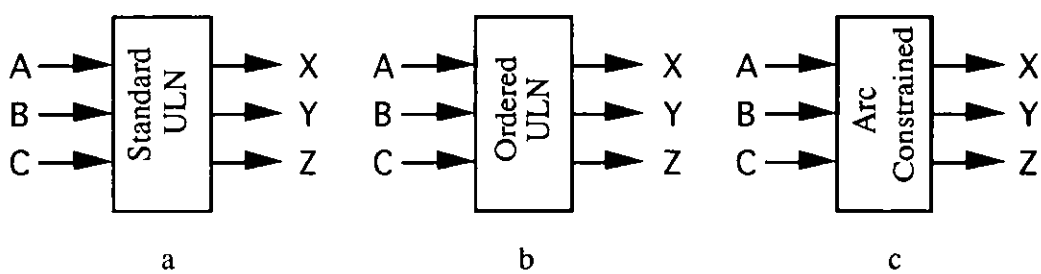


Figure 3.7 Unit Logic Nodes

Although in some cases these node logics can be achieved by using standard Split Node Logics, the ULN provides a much neater representation. Several SNL nodes may be required where one ULN can be used.

An old style Exclusive-Or construct within the ULN is not defined because this logic can be achieved by a deterministic structure within an ordered ULN. For example, if Z←A XOr B, this can be achieved by the ordered ULN:

1. $\overline{Z} \leftarrow A \cap B$
2. $Z \leftarrow A$
3. $Z \leftarrow B$

(i) Standard ULN

Each output arc is linked in a logical way (either And or Blank-Or as in SNL input) to one or more input arcs. For example in figure 3.7a, activity X can be defined to begin when both activities A and B complete successfully and activity Y can be defined to begin when activities A or B or C complete successfully. In this case:

$X = A \cap B$
$Y = A \cup B \cup C$

Example: (all And) In a software development, after completing menu testing successfully and menu selection 'A' coding, this selection can be integrated into the menu driven system. The same is true for menu selection 'B' with the menu driven system and so on.

(ii) Ordered

Only one output arc is performed depending on a preferred ordering applied to each output activity. For example, activity X may be performed instead of activity Y if both their input criteria are satisfied.

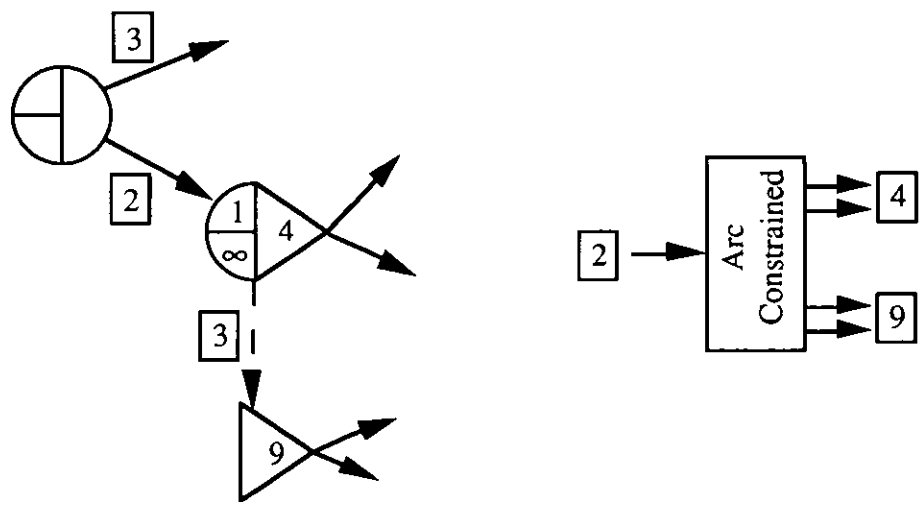
Example: If modules A and B and C fail their user trials it may be necessary to redesign and recode the entire system. This rule would take precedence over 'A and B failing' that could, perhaps, involve debugging their particular modules. Although A and B could prove to be true, A and B and C would take precedence over this rule. This would be represented by the ordered rules:

1. Redesign and Recode = $A \cap B \cap C$
2. Debug A and B = $A \cap B$
2. Debug A = A
4. Debug B = B

(iii) Arc Constrained

This is equivalent to the Filter #3 output defined in VERT [Moeller 1972]. Although not related solely to a node's input arcs, it fits more logically into the ULN definitions. As in the Filter #3 node, each output arc has a list of activities with positive and negative markers. If all activities with positive markers have been completed successfully (their *completed marker* set) and all activities with negative markers have not yet completed (their *completed marker* is not set), that output activity is allowed to start. This is particularly useful for linking activities to events that occurred much earlier in a project life cycle. This approach is much simpler than that adopted by Moore and Clayton (1976). Their technique, called *network modification*, incorporated separate structured routes through a network that would be initialised depending on a single, previously executed activity.

Figure 3.8 shows an example of Arc Constrained network modification where the modified route is represented by a dashed, arrowed line. In this case, if activity 3 were to complete before node 4 had been realised, the system would replace the output from node 4 with node 9. Thus, on completion of activity 2, the outputs from node 9, not node 4, would be initiated. The replacement of node 4 with node 9 only occurs if activity 3 completes before node 4 is realised. If node 4 had already been realised it could not be replaced by node 9 which would remain dormant. The Arc Constrained technique provides a more flexible approach to this problem and allows the incorporation of alternative routes within an initial network. Figure 3.8 (Equivalent ULN) shows this simpler representation.



Arc Constrained Equivalent ULN

Figure 3.8 Arc Constrained and Equivalent ULN

3.2.7 Dummy Activities

To maintain the logic of a network it is sometimes necessary to use *dummy activities*. These activities have no times or costs associated with them and are represented by dashed, arrowed lines in an activity network. They link nodes between which there is a logical dependence, but no actual activity. Dummy activities will be used in examples and definitions used in following sections.

3.2.8 Loops

Another dimension of Generalised Activity Networks is their ability to handle loops. In ordinary Probabilistic and Deterministic activity networks, where all node inputs are deterministic, a loop results in a situation where activities need to complete before they can start. Allowing loops to be formed in an activity network provides useful analysis of projects where certain tasks are repeated, for example software debugging and retesting loops. Loops can be analysed to deduce the expected number of times each loop is performed, with associated times, costs and performance measures. Loops can either be formed by probabilistic branching to earlier stages of a project life cycle, or can be explicitly defined by a specified number of repetitions.

In cases where loops are explicitly defined, it was suggested by Grey (1994) that loops are handled best by repetitive implementation of single activities. This can become rather messy, especially if the loop is to be performed several times. The solution to this problem is an alternative implementation of the Exclusive-Or output from a node. In this case, instead of applying probabilities to the output arcs, a figure is applied to an arc (representing, in this case, the activity looping back) that represents the number of times that activity should be performed. Thus, each time the node is realised, the arc with this applied counter is performed and the counter is reduced. This repeats until the activity has been performed the required number of times, after which the alternative output route from the node is taken. Figure 3.9 shows an explicit loop that will ensure activity A will be repeated four times. This is clearer and simpler than redrawing task a several times, and it also identifies, more explicitly within the network, where repetition is taking place. It is worth noting that loops can involve several activities (for example, activity A in figure 3.9 could represent a subnetwork as could the loop back) in which case repetitive implementation of the loop would be particularly messy.

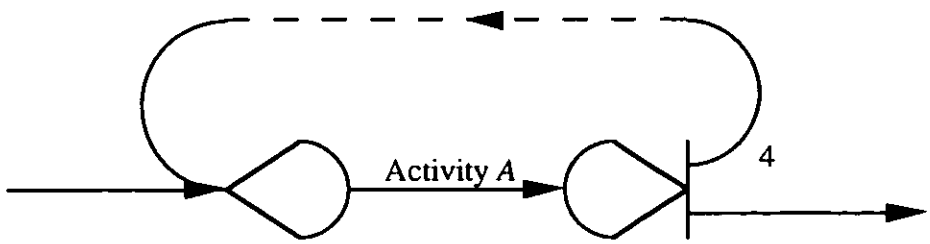
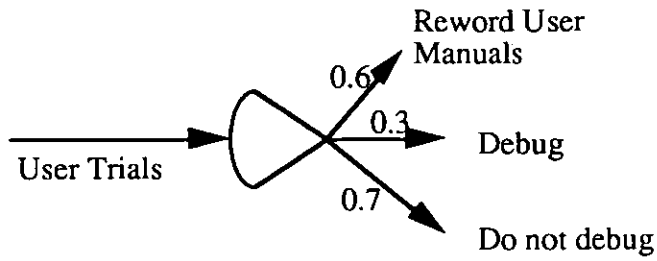


Figure 3.9 An Example Loop

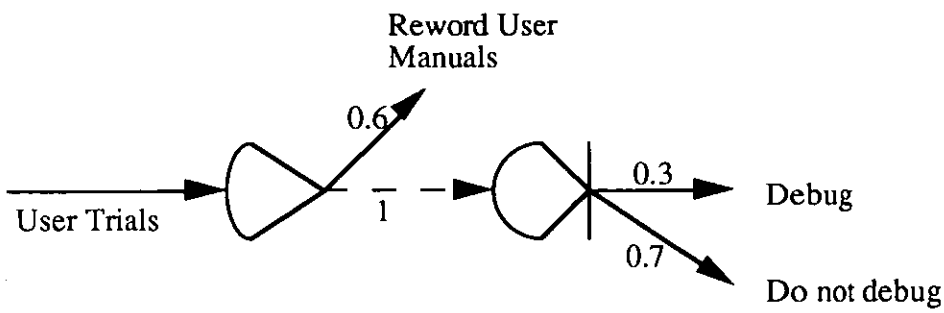
3.2.9 Clarity

With the probabilistic output of SNL nodes defined in five mutually exclusive forms, it is important to differentiate between them within a network diagram. This ensures that the logic and clarity of a network is maintained. For example, after completing user trials in a software development project, it may be necessary to reword the user manuals and/or debug the program. Clearly these tasks are independent from one another but the probability of continuing, without needing to debug the program, is exclusively dependent on having to debug the program. Figure 3.10 shows that the previous representations could not handle clearly this combination of node output styles whereas, by using dummy activities and alternative node representation, the definitions provided in this chapter can. The probabilities assigned to the activities in this diagram are provided as examples.

When a node is the source of a single activity, that node's output should be represented by the deterministic form. If it has an independent probability associated with it, then the Independent-Or form should be used instead. Similarly, a deterministic input should be used when only a single activity enters a node, unless there is a reason why one of the other node inputs are required (for example, if an activity must complete n times before the node is realised).



Previous Representation



New Representation

Figure 3.10 Improved Network Representation

3.2.10 Non-activities

In the previous section dummy activities were used to maintain the logic of a network. Dummy activities can also be used to represent what are called *non-activities* in a Generalised Activity Network. Non-activities are links that represent probabilistic outcomes from certain tasks. For example, non-activities can be used to represent the outcome of a test, whether it is success or failure - figure 3.11. In these cases the non-activities are represented by dummy activities to indicate that they have no cost or time associated with them.

3.2.11 Activity Completion Probabilities

Although not identified in any previous literature in the field, the inherent flexibility of Generalised Activity Networks allows one to apply a completion probability to each activity in a network. The completion probability for activity ij is represented as

p_{ij}(finishes). This probability represents the chances of successfully completing task ij given that it is started. In most cases activities that start do complete successfully, and consequently the completion probability is one (p_{ij}(finishes) = 1) and can be omitted from the network diagram. However, in certain circumstances it is possible that tasks may not complete successfully and completion probabilities are applied to these activities to reflect this. The successful completion of an activity in a project, therefore, depends on the probability that it starts (p_{ij}(starts)) and the probability that it successfully completes (p_{ij}(finishes)). Thus:

$$p_{ij}(\text{completes}) = p_{ij}(\text{starts}) * p_{ij}(\text{finishes}) \qquad (3.5)$$

The standard PERT technique cannot incorporate this form of detail because of the implicit deterministic nature of the technique. Completion probabilities can prove particularly useful for risk analysis where managers need to anticipate possible task failures. For example, in research and development projects one may expect certain avenues of research to fail. Being able to plan for these failures beforehand allows a manager to prepare alternative solutions.

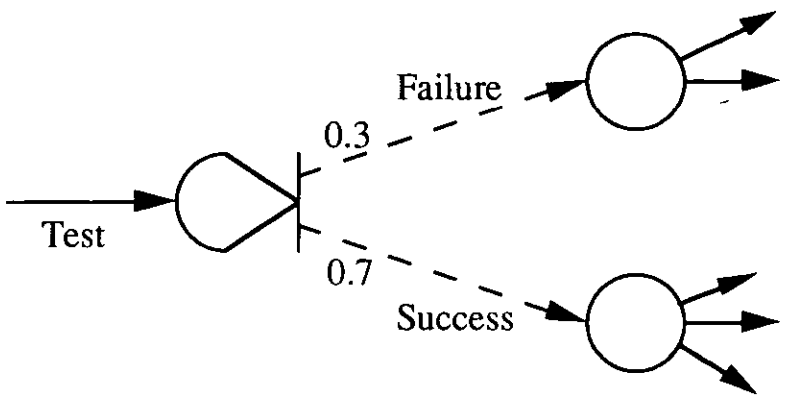


Figure 3.11 Example of Non-Activities

If an activity is deemed to have failed, it is important that the repercussions of this failure are perpetuated through a network. In other words, all other activities relying on the successful completion of this activity must be marked as eliminated from that run of the activity network. Figure 3.12 shows an example of the importance of this effect.

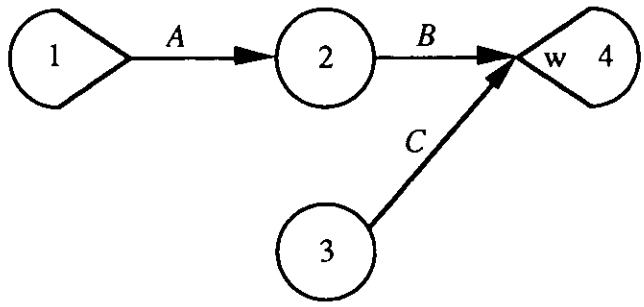


Figure 3.12 Example of Perpetuated Elimination

In this case, if activity *A* fails to start (which is possible due to the probabilistic output nature of node 1), activity *B* has no chance of being performed. If activity *B* was not marked as eliminated, node 4 would be in perpetual limbo as it would be waiting for activity *B* to either be eliminated or completed, which would, of course, never happen. This example emphasises the need to perpetuate failure through a network and mark as eliminated all activities relying on a previously failed task.

3.3 GENERALISED ACTIVITY-ON-THE-NODE NETWORK DEFINITION

3.3.1 Overview

Today, project management software tools are moving away from Activity-on-the-Arrow towards Activity-on-the-Node representations. In Activity-on-the-Node representations, nodes represent activities and arcs represent logical connections between activities. At a recent exhibition in London [Project Management South 1994], out of twelve project management software tools on offer, the majority (ten) provided Activity-on-the-Node representations. MICROPLANNER™ (1992) was the only package offering solely the Activity-on-the-Arrow representation. Pertmaster Advance (1994) was the only tool offering both representations - in the same way as the original Pertmaster product. The ASTA Development Corporation [PowerProject® 1991] had abandoned activity networks altogether and had developed the concept of linked bar charts (Gantt Charts). Linked Bar Charts were on offer with several other packages (for example, Artemis Schedule Publisher™ (1993)) but these packages provided Activity-on-the-Node representations as well. The general mood was one of a movement away from older Activity-on-the-Arrow representations towards standardised Activity-on-the-Node and Gantt charts (often referred to as Bar charts) based Windows™ products. With this direction in mind, it was important to define a Generalised Activity-on-the-Node Network to reflect the direction in which most project managers today are heading.

Another advantage of an Activity-on-the-Node representation is the ease in which precedence diagram constraints (also known as logical dependencies constraints) can be applied (for example, finish to start, start to finish constraints). In Activity-on-the-Arrow representations, applying these constraints involves the introduction of both dummy activities and nodes between activities being constrained [Crandall 1973]. Trying to apply these constraints in Generalised Activity-on-the-Arrow Networks is even more complex as dummy activities must be introduced alongside new nodes with specific characteristics that maintain the original logic of a network. Applying logical dependencies to Generalised Activity-on-the-Node Networks is much easier as the constraint can be applied directly to the logic connections linking activities.

Turner (1993) also identified several other reasons why the Activity-on-the-Node representation was preferred by many managers to Activity-on-the-Arrow.

1. Work is more naturally associated with a box.
2. It is more flexible for drawing networks.
3. Software is easier to write for this representation.
4. Gantt charts are easier to draw, requiring no dummy activities
5. The work is independent of the logic which can be added later.

One difference between Generalised Activity-on-the-Arrow and Generalised Activity-on-the-Node representations is that, in the latter, nodes in a diagram are stretched to accommodate any necessary activity identifying text. Figure 3.13 shows an example of a typical node. In this case activity *A* has a deterministic input and probabilistic output.

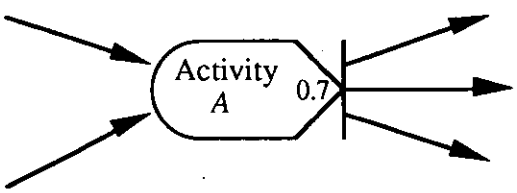


Figure 3.13 An Example Activity in a Generalised-Activity-on-the-Node Representation

Because this definition of a Generalised Activity Network representation contains both Unit Logic Nodes and Dummy Nodes it can, in some respects, be viewed as a Hybrid of Activity-on-the-Arrow and Activity-on-the-Node. For the purposes of this thesis, however, it will be referred to as a Generalised Activity-on-the-Node representation.

3.3.2 Activity Node Inputs

The development of a Generalised Activity-on-the-Node Network leads directly from the node definitions of the Generalised Activity-on-the-Arrow Network defined earlier in this chapter. As the definitions represent all node input logics, they can be used directly as definitions of Activity Node inputs in Generalised Activity-on-the-Node Networks. The node input definitions, And, Or and XOr are exactly the same as those defined earlier in this chapter with the definitions relating to logic relationships between activities.

(ii) And

All activities preceding an activity must be completed before that activity can be performed.

(ii) Or

This activity requires n preceding activities to complete before it can be performed. This definition implicitly accepts that the activity can be performed several times, each time n preceding activities complete. As before, when n=1 this can be omitted from the node. The different codes applied to the input of the activity affect this reasoning in the same way as the Activity-on-the-Arrow definition earlier:

H = *Blank* - as for the above definition. It is not necessary to wait for the activity to finish before starting it again (like code Q dictates).

$H = C$ - Allow all other activities preceding this activity to complete in their own time. The activity will be performed once only.

$H = H$ - Cancel all remaining, active preceding activities when the activity starts and eliminate all others.

$H = W$ - The activity is performed when all active, preceding activities are completed and all other preceding activities have been eliminated. The node still requires a minimum of n activities to complete successfully before realisation, and can be realised only once.

$H = Q$ - This represents a queuing activity that can only deal with n completing activities at any one time. The activity must complete before allowing the next n activities to restart the activity again.

(iii) XOr

This works in the same way as the Activity-on-the-Arrow definition and ensures that n

different activities complete successfully before applying the above rules.

3.3.3 Activity Node Outputs

Again, the definitions of activity output logics are derived directly from the Activity-on-the-Arrow definitions provided earlier.

(i) Deterministic
All logical relations emanating from this activity will be pursued when the activity completes. This is the standard Activity-on-the-Node representation found in almost all project management software tools.

(ii) Independent-Or
All logical relations from this activity will be pursued, depending on their probability values, when the activity completes. In a similar way to the Generalised-Activity-on-the-Arrow representation this can lead to a situation where all or, at the other extreme, no logical relations are pursued when the activity completes.

(iii) Exclusive-Or
In this case only one logical relation leaving the activity will be pursued when the activity completes. The probabilities, rules, and examples are the same as those in the Generalised-Activity-on-the-Arrow representation.

(iv) Dependent-Or
The logic relations with other activities that emanate from this form of activity output are in some way dependent on one another. As for the independent case, each logic relation has its own probability of occurring, but the combined probability no longer equates to the product of the individual activity occurrence probabilities.

(v) Cost/Time Dependency
The logic relations emanating from this activity are dependent on the times and costs accrued to date. This is exactly the same as the Generalised Activity-on-the-Arrow representation.

3.3.4 Activity Completion Probabilities

Activity completion probabilities (such as those explained earlier) can be applied directly to the output side of an activity node in this Generalised Activity-on-the-Node representation. This value represents the probability that the activity completes successfully given that it starts.

Figure 3.13 brings together the points introduced above, showing how activities can be represented with different input and output forms and illustrating an example of an applied completion probability (in this case p_A(finishes)= 0.7).

One of the strengths of the Generalised Activity-on-the-Node representation defined in this chapter is its similarity to conventional Deterministic and Probabilistic Activity Network styles. Figure 3.14 illustrates how an activity network, generated using these rules, can look virtually identical to a standard probabilistic activity network representation, when the more powerful Generalised Activity-on-the-Node functions are not required. With this similarity to more popular techniques, it requires no retraining at the initial stages of its implementation. Managers can be introduced to new concepts and representations provided by this Generalised Activity-on-the-Node representation in a progressive way.

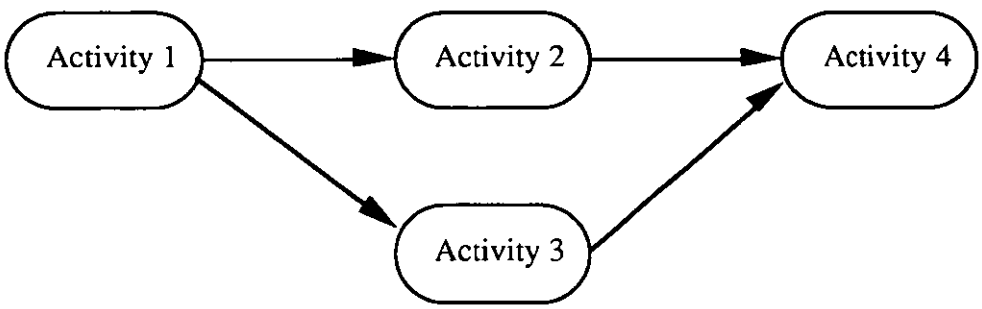


Figure 3.14 A Simple Generalised-Activity-on-the-Node Network

3.3.5 Unit Logic Nodes

Unit Logic Nodes provide an immense amount of power to Generalised Activity-on-the-Node Networks by providing a means for filtering future routes through a network. They determine which subsequent activities should be performed, based on what has happened previously. All three ULN nodes, defined earlier in this chapter, can be used in a Generalised Activity-on-the-Node Network. Figure 3.7 shows the three node types available for Generalised Activity-on-the-Node Networks. The various incoming and outgoing arcs represent logic relationships with other activities in this case. Apart from milestones and dummy nodes, Unit Logic Nodes represent the only other nodes in a Generalised Activity-on-the-Node Network where resources are not consumed.

(i) Standard ULN

Each output logical relation is linked in a logical way to each incoming logical relation. The example provided earlier illustrates this situation. In this case the example relates directly to logical relations to subsequent nodes.

(ii) Ordered

Only one output logical relation is pursued depending on a preferred ordering applied to each output logical relation. Again, the example provided earlier illustrates the possibilities of this function.

(iii) Arc Constrained

Exactly as in the Generalised Activity-on-the-Arrow representation, each output logical relation from this node has a list of activities with positive and negative markers. If all the activities with positive markers have been completed successfully (their *completed marker* set) and all the activities with negative markers have not yet completed, that output logical relation is allowed to be pursued.

3.3.6 Dummy Nodes

It may be necessary to introduce dummy nodes into a Generalised Activity-on-the-Node Network to preserve the logic of a connection. As an example, suppose, from the outcome of a test (which is either successful or not), two possible activities may need performing with independent probabilities. Trying to incorporate both an Exclusive-Or output (the test result) and an Independent-Or output for these activities, onto the output side of the Test node is rather messy. The solution is to use a dummy node that enables the logic to be maintained. Figure 3.15 shows how a dummy node is used in this situation. To ensure that there is a connection between the dummy node and activity *C* (it is not guaranteed by Activity *A* and *B* which are probabilistic) a logical relation between these nodes has also been applied with probability one. The other probabilities are provided as examples.

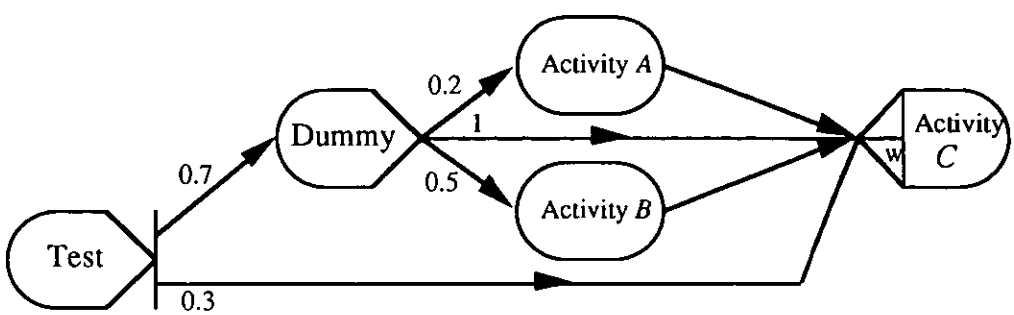


Figure 3.15 Example of Dummy Node Usage

3.3.7 Loops

Loops are formed as easily in Generalised Activity-on-the-Node networks as they are in Generalised Activity-on-the-Arrow networks. Figure 3.16 illustrates how an activity, *A*, that requires performing four times, would be represented. The similarities between this representation and that shown in figure 3.9 are self evident. The Exclusive-Or output is being used in the same way in this example as it was earlier. In this case the output defines that the logical relation leading back should be performed four times - each time activity *A* completes. Loops can also be produced by probabilistic branching in which cases probabilistic activity outputs would be used and loops would only be performed by chance. Unit Logic Nodes can also be used to initiate feedback to earlier stages of a project plan. In this case they provide an incredible amount of planning power as feedback can be dependent on events that occurred earlier in a project. An example of this will be presented later.

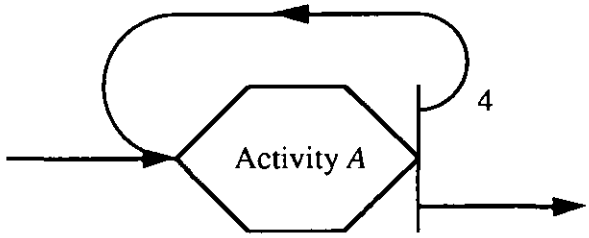


Figure 3.16 Example of a Loop in a Generalised Activity-on-the-Node Network

3.3.8 Non-Activities

Non-activities, that are used in Generalised Activity-on-the-Arrow Networks, are not required in the Generalised Activity-on-the-Node representation as they are represented by the logical relationships between activities. For example, figure 3.11 can be represented in a Generalised Activity-on-the-Node representation as shown in figure 3.17. The logical connections between activities in this case replace the need for non-activities in this form of activity network. It is worth noting that the power of this Activity-on-the-Node representation also allows a completion probability to be applied simultaneously to an activity. In figure 3.17 the probability of actually completing the test itself is 0.9, the probability of the test proving successful is 0.7, and failing is 0.3. In this case the probability of performing the Success Activity is 0.63 (the product of successfully completing the test and the test proving positive).

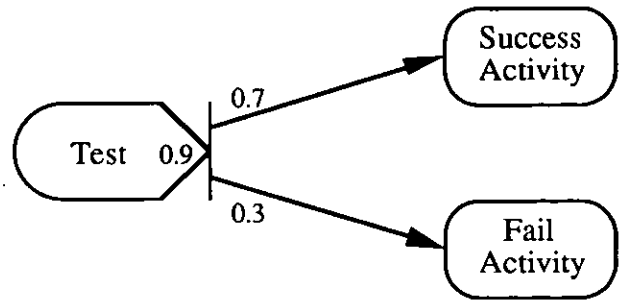


Figure 3.17 Completion Probabilities and Probabilistic Branching

3.3.9 Precedence Diagram Method (PDM)

PDM constraints (or logical dependency constraints) have being used by project managers for a long time [Crandall 1973, Wiest 1981, Moder and Phillips 1983]. They have also been successfully implemented into several current project management tools for example, Microsoft® Project 4.0™ (1994), Project Manager Workbench™ for Windows™ (1994), and Primavera Project Planner® (1994). It was noted earlier in this chapter how such constraints are difficult to implement within Activity-on-the-Arrow networks. This is not the case with Generalised Activity-on-the-Node Networks that prove to be as open to his approach as standard Probabilistic Activity-on-the-Node Networks.

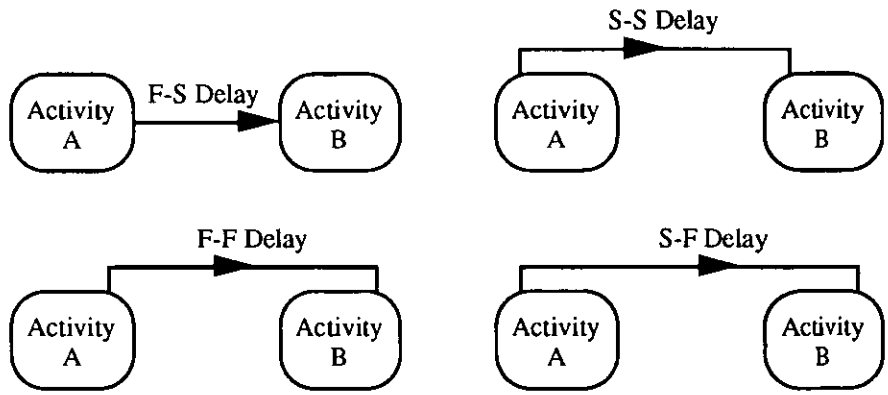


Figure 3.18 The Four Possible PDM Constraints

The four possible PDM constraints that can be applied between any two activities are represented in figure 3.18. The definition of these constraints was covered in more detail in chapter one. Applying these constraints to Generalised Activity-on-the-Node networks could not be easier as they are applied directly to the logical relations that connect any two nodes. Figure 3.19 shows a simple example of a Generalised Activity-

on-the-Node network with some PDM constraints applied. In this example there are Finish-to-Start constraints between activities *C* and *D*, and between *E* and *F*. There is a Start-to-Start constraint between activities *A* and *C*, and a Start-to-Finish constraint between activities *A* and *B* (although the use of such a constraint is rather limited).

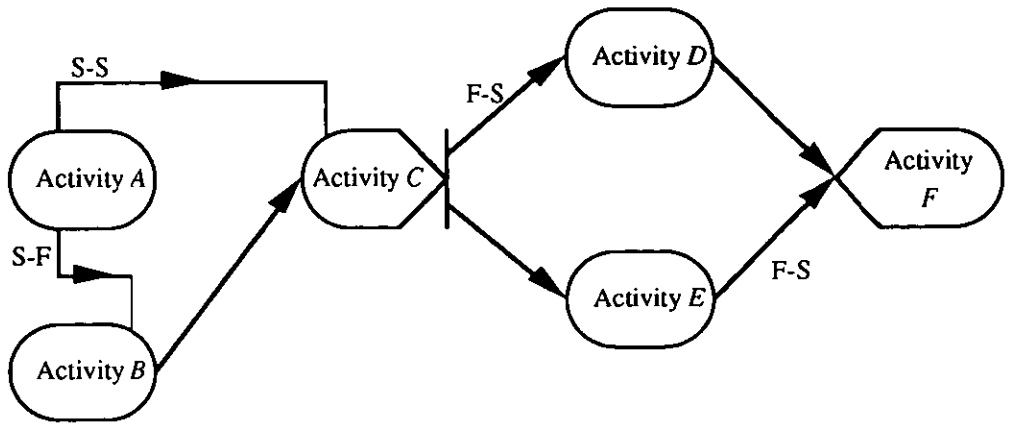


Figure 3.19 An Example Network with PDM Constraints

3.3.10 Milestones

A project milestone is a moment that represents a significant step towards the completion of a project. Determining a project's milestones beforehand helps to define goals for the project and maintain the developers awareness of those goals. They also give a visibility to the progress of the project in a measurable sense and highlight any problems in keeping to schedule, that in turn allows remedial action to be taken at the earliest opportunity. Examples of milestones include:

The project start day.
The delivery of a new computer.
Part of the system is ready for demonstration.
Part of the system is delivered to the customer.
The final system is delivered.

Representing milestones in Generalised Activity-on-the-Node networks is not as clear cut as it might first seem. In ordinary Activity-on-the-Node Networks it can be a simple case of representing a milestone in a rounded box or oval (for example, figure 3.20). In Generalised Activity-on-the-Node Networks the input and output logics to the milestone may need preserving, in which case it is necessary to maintain the logical style of the node. The solution in Generalised Activity-on-the-Node networks is to represent milestones in dashed nodes that allow the logic of a node to be maintained. Figure 3.21

provides a simple example of milestone representation in a Generalised Activity-on-the-Node network.

Figure 3.20 Milestone Representation in an Activity-on-the-Node Network

Figure 3.21 Milestone Representation in a Generalised Activity-on-the-Node Network

3.4 APPLICATIONS

3.4.1 Overview

Virtually all project management software in use today is based on either the standard Deterministic or Probabilistic Activity Network solution. It was noted by Kidd in 1990 [Kidd 1990] that 'these programs cannot address the strategic uncertainties that face all project managers.' Kidd went on to identify that a solution would be for managers to use more sophisticated management tools such as the VERT technique mentioned earlier in this chapter. Unfortunately, as noted earlier, the VERT technique represents too broad a change for managers to make. The alternative is the implementation of a technique that has much in common with the popular PERT technique, with the added flexibility and clarity available as and when it is required. The completion of both Generalised Activity-on-the-Arrow and Node Network representations provided in this chapter provides just such a definition.

The flexibility offered by Generalised Activity Networks enable them to be used in all kinds of development projects. In their basic form they can be implemented as purely deterministic network structures. In this configuration they represent common Probabilistic Activity Networks and can be used in the same way as the standard PERT technique. Where Generalised Activity Networks come into their own is in projects where uncertainties exist and conclusions of particular tasks cannot be predetermined. An obvious field where these problems are encountered is Research and Development where outcomes from various research activities can only be guessed at initially.

Another field is Software Development where projects include test results and integration and implementation uncertainties that cannot be fully guaranteed at a project's outset. Being able to code these uncertainties into a project plan at the outset enables mangers to perform a more detailed risk analysis.

In previous literature, examples of Generalised Activity Networks mentioned earlier are given in specific, real life projects. For example, Moeller (1972) shows how VERT was used in the planning of a helicopter development. Moore and Clayton (1976) shows the GERT technique being used in planning the drilling of an oil well, and Kidd (1990) provides an example of how a Generalised Activity Network can be used in managing a software development project.

3.4.2 An Example Problem

In order to see how both the Generalised Activity-on-the-Arrow and Generalised Activity-on-the-Node Networks defined in this chapter could be used in the planning of a project, the following example, based on a simple software development, has been devised. This example is related to the design and implementation phases of a project - representing a more tactical level plan than the strategic metamodel level. An example of a Generalised Activity Network for a metamodel can be found in chapter two. It is unlikely that any one project would require all possible Generalised Activity Network functions defined in this chapter. To try and incorporate every function into an example would prove too complex for illustrative purposes. However, as shown in previous examples, each logic has its practical uses and managers would employ any required subset of the logics available. The example, that helps to illustrate the possibilities of Generalised Activity Network usage, is based on the following scenario:

A software system to be developed consists of three modules - A, B and C. These modules are coded and tested separately after the design is complete. If all modules fail their testing, the whole system must be redesigned and recoded as major problems have clearly been encountered. Although module C must be working before the project can continue, any bugs found within modules A and B can be ignored as they will be removed automatically during the integration phase. The user manuals can also be written concurrently with the integration of modules A, B and C.

After integrating the modules and completing the user manuals, the viability of the project will be assessed. If, at that stage, the project is overrunning its expected costs by 50% or more the project must be scrapped. Otherwise the project continues with the integrated testing phase. It is anticipated that the only reasons for this test failing are due to the system being too slow or too costly (in terms of memory requirements).

These problems need rectifying before user trials can take place. The only problems anticipated with the user trials are that the manuals are inadequate, or some simple bugs are found resulting in a basic debugging process. No further testing will take place after this phase and this implementation stage of the project is completed.

(i) A Solution Using PERT

To begin, an attempt is made to plan this project using a standard PERT type approach (both Activity-on-the-Arrow and Activity-on-the-Node). The PERT network is somewhat difficult to form from this scenario due to the uncertain behaviour and repetition of particular tasks. The resultant PERT networks are shown in figures 3.22 (Activity-on-the-Arrow) and 3.23 (Activity-on-the-Node). Uncertainties at various stages of the project cycle have had to be implicitly coded into specific tasks in these networks. For example, *Improvements 2*, following *User Trials*, includes the possibility of having a zero cost/duration or a cost/duration that is based on recoding some software or rewriting parts of the user manual. The deterministic structure of PERT has not allowed these alternatives to be explicitly incorporated into the plan and they have become buried in the simplicity of the technique.

(ii) A Solution Using a Generalised Activity-on-the-Arrow Network

The Generalised Activity-on-the-Arrow Network, on the other hand, allows more detailed analysis of the project plan, explicitly identifying problems that may occur. The Generalised Activity-on-the-Arrow Network, based on this project scenario, is shown in figure 3.24. In this diagram, all uncertainties are shown explicitly, providing the project manager with a more detailed view of possible risk points in the project life cycle. Each node in figure 3.24 has been assigned a number to assist with the explanation below.

It is worth noting how the different node input styles have been used in constructing this network. While the deterministic node input at node 9 should be clear it is worth explaining the W-Or node inputs at nodes 13 and 17. The W-Or inputs at these nodes ensure that the project only progresses when all active incoming activities to the node complete successfully. It would be wrong, for example, to continue with the *User Trials* (between nodes 14 and 15) while attempts were still being made to increase the software speed, even though a reduction in the memory costs had been achieved.

Because of the probabilistic nature of the activities between nodes 12, 13 and 16, 17 (ie it is possible that none of these activities may be performed), a dummy activity also links these nodes. This ensures that the project plan will not grind to a halt should either of these pairs of activities be eliminated from the project plan.

Nodes 14 and 18 represent Blank-Or inputs (an W-Or input could have been used here

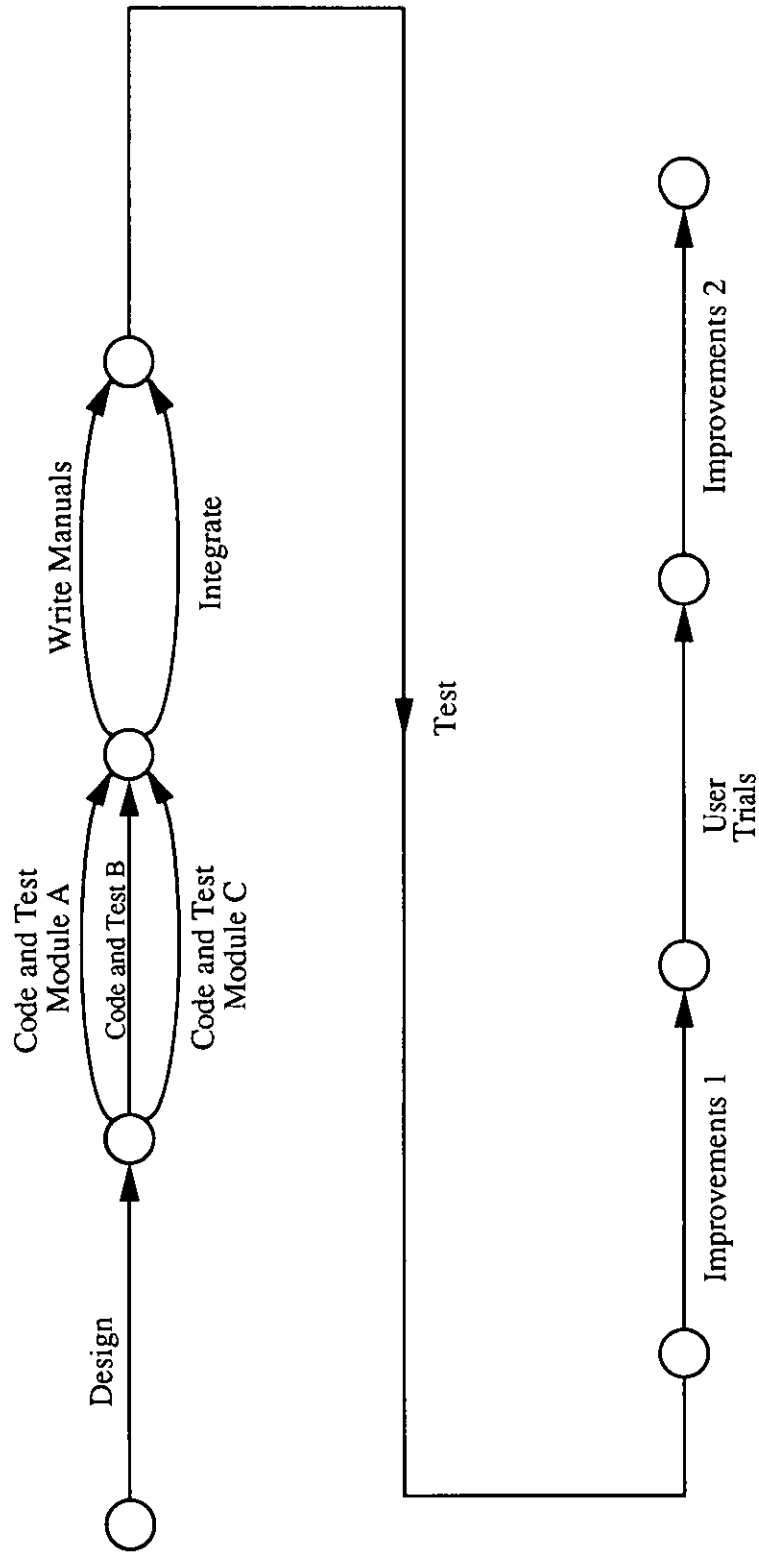


Figure 3.22 Probabilistic Activity-on-the-Arrow Network of Example Project

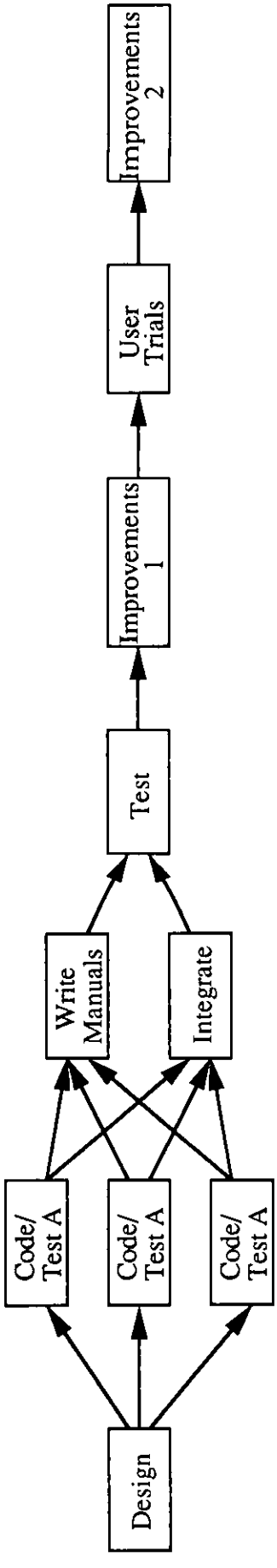


Figure 3.23 Probabilistic Activity-on-the-Node Network of Example Project

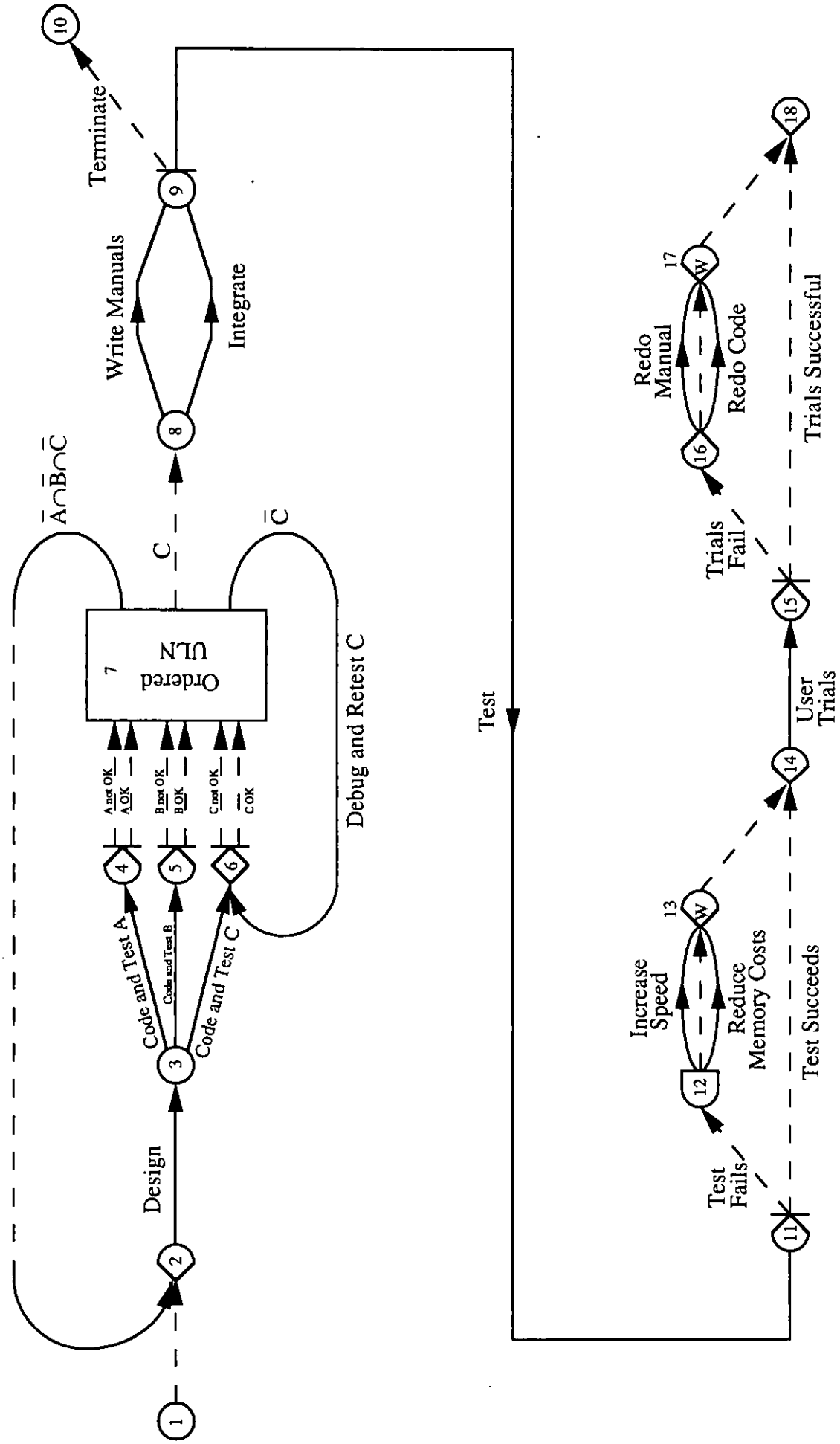


Figure 3.24 Generalised Activity-on-the-Arrow Network of Example Project

as well). Nodes 2 and 6 also represent a Blank-Or input. Node 6 allows the result of testing module C to be passed straight on to the Unit Logic Node (node 7). This ensures that node 6 can be realised several times due to the loop consisting of the *Debug* and *Retest* of module C from the ULN.

The output activities from the Ordered ULN are based on the following rules that are coded into the ordered logic of the node.

1. $\bar{A} \cap \bar{B} \cap \bar{C}$
2. $\bar{C}$
3. C

The node output logics at nodes 4, 5, 6, 8, 9, 12 and 16 clearly match the project scenario as detailed.

It should be noted that, as it stands, this Generalised Activity-on-the-Arrow Network provides no temporal or cost information to the project manager. However, it does provide an initial visual representation of the project and identifies possible risk points and repetition in the project life cycle. In order to be of some more analytical use the activities are assigned duration/cost functions and probabilities, and the network is analysed. How networks such as these are analysed is discussed in the next chapter.

(iii) A Solution Using a Generalised Activity-on-the-Node Network

Figure 3.25 provides a Generalised Activity-on-the-Node representation of the project scenario detailed earlier. As one would expect, there are many similarities with the node input and output logics of the Activity-on-the-Arrow network discussed above. Points worth noting in this representation are the inclusion of milestones, representing the start (M1) and completion (M3) of this part of the project. M2 represents the terminal completion of the project if it has overrun its budget by the amount specified earlier.

The coding and testing of module C has had to be separated to allow the feedback from *Debug C* to feed into the testing stage. Without this separation the feedback would have lead into the coding of module C which is not the case. Dummy nodes have also been included after the integrated modules have been tested and after the user trials. These ensure the logic of the scenario is maintained and do away with the difficulty of applying two different probabilistic output types to the *test* and *user trials* nodes.

The ULN is coded in exactly the same way as the Generalised Activity-on-the-Arrow description, and all other node inputs and outputs are the same.

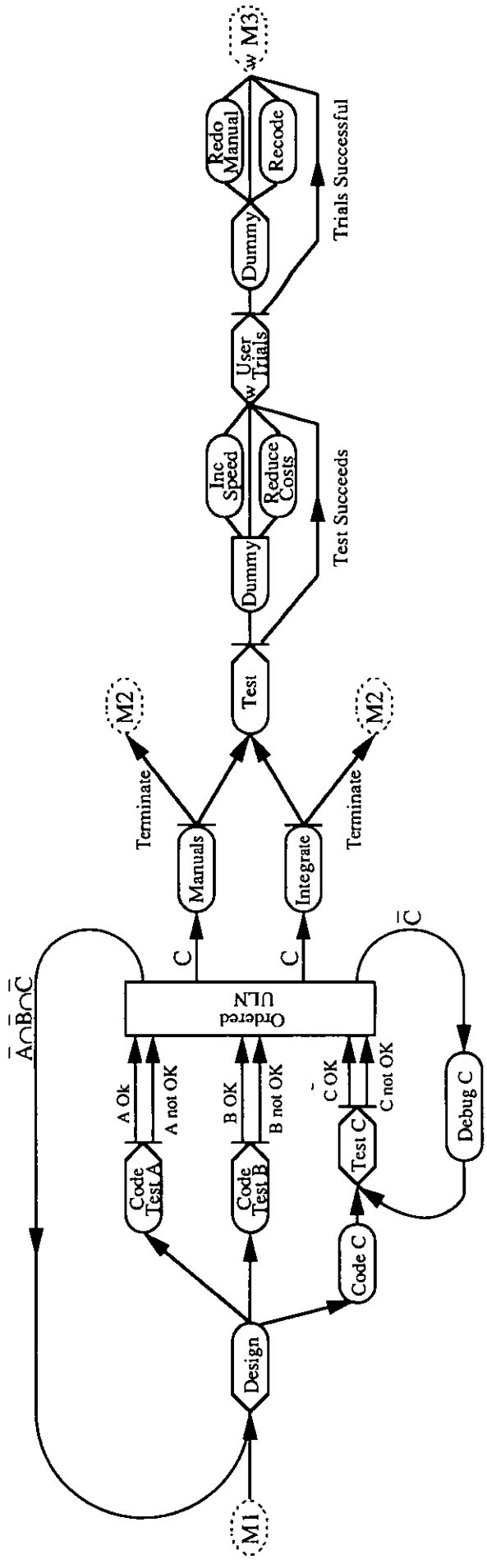


Figure 3.25 Generalised Activity-on-the-Node Network of Example Project

From these three representations of the project scenario the PERT technique was the weakest being unable to represent all the possibilities in the project scenario explicitly. The two Generalised Activity Network versions provided a more accurate representation of the project scenario and, of the two, the Generalised Activity-on-the-Node representation is perhaps the clearer. This representation also has the advantage of providing PDM constraints when they are required and, as the mood of the project management industry has indicated, represents the most popular activity network representation these days.

3.5 NETWORK PROPERTIES

3.5.1 Overview

Parameters are applied to tasks within activity networks to provide measures by which projects can be planned and controlled. The majority of work is directed towards time and cost values, although performance measures [Moeller and Digman 1981] can also be used. Although many project managers treat time and project management as synonymous [Turner 1993], time is not the only variable within a project. Activity networks should be used to manage both times and costs within a project providing a means of optimising and controlling the interaction between these factors. The interaction of resources (that also equate with costs) and temporal factors within Generalised Activity Networks is a particularly complex task. No work to date has been noted in this area and it proves to be an interesting area for future research. For the purposes of this thesis, the affect of temporal factors (the more complex of cost and time factors) is assessed in chapter five.

3.5.2 Durations

Various distribution functions (referred to as *temporal functions*) have been proposed to represent the duration of activities in Generalised Activity Networks. These temporal functions include the usual PERT three time Beta distribution estimate, single time estimates and so on. In literature on the subject, one of the most flexible methods proposed for applying these functions to activities was that proposed by Moore and Clayton (1976) for the GERT technique. Their technique provides ten possible function estimates that are listed below. In their technique each activity was associated with three parameters (shown in figure 3.26).

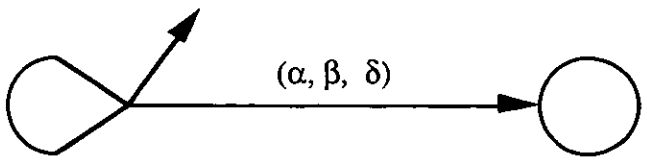


Figure 3.26 GERT Time Representation

α The probability that the branch will be taken ($1 \geq \alpha > 0$). Clearly $\alpha = 1$ when the originating node is deterministic.

β A reference to a parameter set where the data associated with the distribution type is stored (figure 3.27).

δ A code identifying the temporal function. The ten possible distribution codes are:

1 A constant value
2 Normal distribution
3 Uniform distribution
4 Erlang distribution
5 Lognormal distribution
6 Poisson distribution
7 Beta distribution
8 Gamma distribution
9 Beta fitted to three parameters (as in PERT)
10 Triangular distribution

Figure 3.27 provides an example of how an activity, with probability 0.2 of occurring and Normally distributed temporal function, is represented by relating the arc to a corresponding reference parameter set associated with the activity.

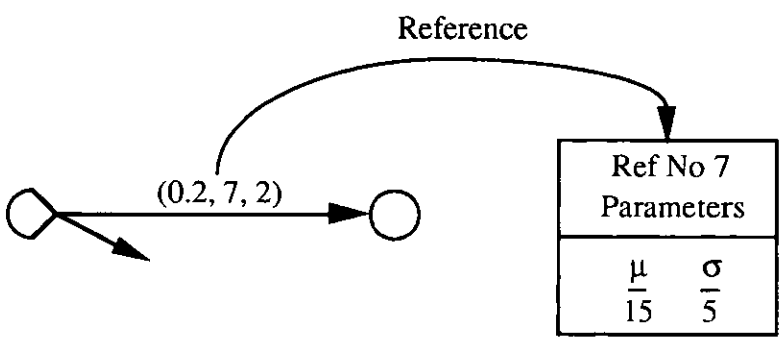


Figure 3.27 More Detailed GERT Time Representation

In more recent project management packages, several other distribution functions have been made available to the project manager. For example, Predict!™ (1992) - a Probabilistic Activity-on-the-Node software tool developed by Risk Decisions Limited - provides twenty one different temporal functions that can be applied to project activities. The sales team for this product point out that it is very unlikely that any one manager would require all of these functions, using a subset of the more popular ones most of the time. @Risk™ (1990) is an add-in to Microsoft Excel™ that allows probabilistic risk to be determined within an Excel spreadsheet. In addition to the ten distribution functions provided in GERT above, @Risk™ also provides:

Binomial	Chi-square	Correlations
Cumulative	Discrete	Discrete Uniform
Error Function	Exponential	General
Geometric	Histogram	Hypergeometric
Logistic	Negative Binomial	Pareto
Weibull		

In addition, Predict!™ provides many of these functions along with:

Bernoulli	Cauchy	F-distribution
Sensitivity	T-distribution	

The affect that some of these activity temporal functions have on the duration of an activity network is studied in some detail in chapter five. The effectiveness and implementation of these activity temporal functions is an interesting area for future research.

Risnet™ (the risk analysis software tool mentioned earlier) provides twelve activity temporal functions including an Exponential-Triangular and a Triangular-Exponential. Monte Carlo™ 2.0, the other risk analysis package available today, provides ten such functions.

3.5.3 Resources

Before a project baseline can be set it is necessary to determine the constraints imposed on that project. Resources represent the most common of these constraints [Turner 1993]. If adequate resources are not available for a project (which, according to most managers, is usually the case) it can cause that project to be adversely delayed. Also, if particular resources are over or underused at any stage (for example, machine hire which could be over or under booked, staff overtime costs and so on) this can add unnecessary

costs to a project.

It is also possible that resources are not available constantly during the life cycle of a project and the effects of this must be anticipated. It is also possible that resources are needed by activities non-uniformly. There are different ways in which activities can be applied to activities in a project - represented by a resource profile for each activity. Turner (1993) identifies four such ways of applying resources to activities - constant, stepped, triangular and Normal. Project Manager Workbench™ (1994) is an example of a project management tool that allows managers to apply different resource profiles to activities. It also allows resource profiles to be considered from a more global project viewpoint and considers these factors when scheduling a project.

Scheduling a project to optimise resource usage can prove to be particularly difficult and several algorithms can be used to achieve this. Examples include Berman (1964), Burgess and Killebrew (1962), Clark (1961a), Levy et alia (1962), King (1964), and Davis (1974) which are mentioned in chapter four.

How Generalised Activity Networks schedule projects with various constraints poses some interesting problems. For example, should a machine be booked for an activity with only 70% chance of been performed? The scheduling of Generalised Activity Networks with various constraints is clearly an area that needs addressing in future research and is beyond the scope of this thesis.

3.6 CHAPTER SUMMARY

3.6.1 Overview

This chapter has provided a conclusive definition for both Generalised Activity-on-the-Arrow Networks (based on the work presented in Dawson and Dawson (1994a)) and Generalised Activity-on-the-Node Networks.

3.6.2 Conclusion

Previous research in this field has been sadly lacking and implementation of Generalised Activity Network' ideas even less so. The majority of popular software planning tools available (for example, Project Manager Workbench™ (1994), CA-SuperProject® (1993)) are based on the ideas of Deterministic Activity Network structures. A few, for example Predict!™ (1992), Artemis Schedule Publisher™ (1993), Primavera Project Planner® (1994) with the Monte Carlo™ 2.0 (1993) package,

provide more risk analysis by implementing the ideas of Probabilistic Activity Networks and providing several activity cost and temporal functions. Only GERT and VERT instigated attempts to implement Generalised Activity Networks. Since these appeared in the late 1960s and 1970s, before the explosion of project management tools, they have been somewhat overlooked by managers of today. Only two tools - Risnet™ and Monte Carlo™ 2.0 provide any means of managing project uncertainties and even they provide only limited functionality.

The future for the work presented in this chapter is the implementation of the definitions provided for a Generalised Activity-on-the-Node representation. The Activity-on-the-Node representation not only provides more clear network diagrams and includes PDM constraints but it also reflects the current mood of the industry and provides a more powerful means of project planning and control.

3.6.3 Future Work

Four areas for future research have been identified from this work:

- Implementation of the definitions in a software tool
- Gantt chart representation of Generalised Activity Networks
- Cost/Time optimisation and resource levelling in Generalised Activity Networks
- Developing a Work Breakdown Structure for Generalised Activity Networks

Whether it is possible to implement a Generalised Activity Network in a hybrid Gantt representation is an area that requires some thought. It is perhaps unlikely that combining these two charts into one would provide any real benefit as the resultant graph would be overly complex. How probabilistic branching is used within a Gantt chart and how it affects scheduling is also an area that requires addressing. The most interesting problem is that posed by the scheduling of Generalised Activity Networks. Various heuristics require developing and applying to this particular problem area. What resources should be assigned to an activity, that may or may not occur, and how time/cost/resource tradeoffs could be performed between probabilistic activities is an interesting problem. How a Work Breakdown Structure should be performed and represented in the implementation of Generalised Activity Networks is also an area for future research.

CHAPTER 4

Project Management and Activity Networks

CHAPTER PREFACE

This chapter deals with techniques for the temporal analysis of both Probabilistic and Generalised Activity Networks. In order to put this work into context the chapter begins by introducing the concepts of project management and previous computational approaches, before moving on to Monte Carlo simulation of activity networks. An improved technique for limiting the number of simulations required in activity network analysis is developed (presented in Dawson and Dawson (1993b)). Antithetic variables are also used to improve the efficiency of this method. A Modal Class simulation algorithm is also introduced (presented in Dawson and Dawson (1993b and 1994b)).

CHAPTER KEYWORDS

Project Management, Monte Carlo Simulation, Dynamic Sampling Technique, Antithetic Variables, Modal Class

4.1 INTRODUCTION

This chapter concentrates on the temporal analysis of activity networks using a Monte Carlo simulation technique. In order to put this analysis into perspective, the following sections introduce the concepts of project-based management and identify the area in which this analysis is used.

4.1.1 Project-Based Management

A traditional, established approach for the management of projects focused on three particular objectives - time, cost and quality. It was felt that if these three objectives could be achieved by good management and optimum tradeoffs between each of them, projects would be performed successfully. Turner (1993) showed that to deliver a project successfully, two other objectives must also be managed - scope and organisation. Figure 4.1 (taken from Turner (1993)) shows how all these five factors

are integrated through the organisational objective. The boxes highlight the techniques that are used in meeting particular project objectives, identified in the circles. This chapter cannot hope to cover all of these objectives at all different project levels, therefore it concentrates on an analytical approach that assists the management of time objectives at lower organisational levels.

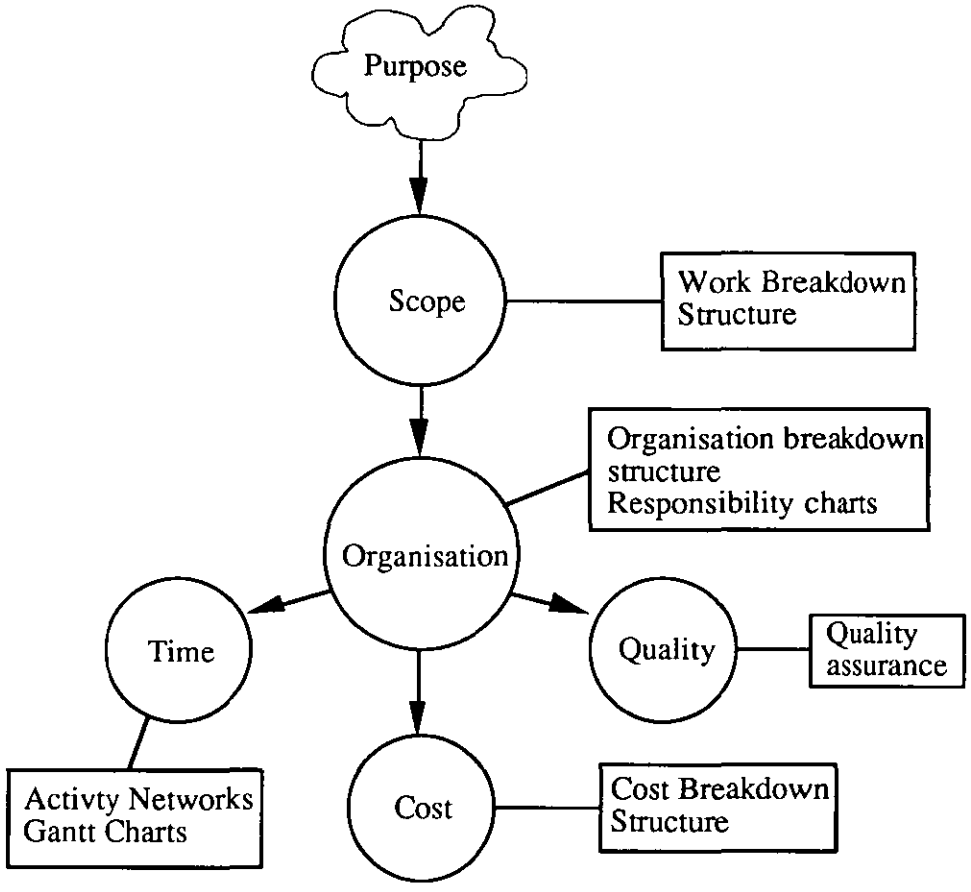


Figure 4.1 Turner's Structured Approach to Project Management

Once the requirement for a project has been identified (it has a purpose), management needs to perform a series of functions to develop an idea (a plan) of how that project will be tackled (this takes place within the analysis phase). The methodology shown in figure 4.1 is an ideal representation of how projects should be managed. In reality a common methodology used by many managers today, for managing smaller projects, usually includes the following set of component functions at some level (a larger project management methodology is covered in section 4.1.9):

(i) Work Breakdown Structure.
(ii) Identification of milestones.

(iii) Activity precedence.
(iv) Application of cost, resource, time, performance estimates.
(v) Calculation of expected costs, times, performance requirements of a project.
(vi) Optimisation/Scheduling.
(vii) Control.

These components are identified, to some degree, in numerous texts including Howes (1984), Pressman (1994), Sommerville (1993), and Plasket (1986) to name but a few.

4.1.2 Work Breakdown Structure

Developing a Work Breakdown Structure is an established technique and is covered in several texts including Tausworthe (1980), Howes (1984), Morreale (1985), and Plasket (1986). More modern approaches to the decomposition of projects include work by Wilson and Sifer (1988 and 1990).

According to Turner (1993) a common pitfall in planning projects is to plan them at a detailed level only. Turner went on to point out that this is a common problem within the development of software systems. Projects cannot be planned at just one level within a business as they need justification at all levels of an organisation's structure. There must also be some coordination between a project at different levels. Turner identified three fundamental levels at which the five objectives identified earlier need to be managed - *integrative*, *strategic*, and *tactical*. At the integrative level, the scope of a project must be identified and its purpose must be within the bounds of the organisation's objectives (figure 4.2). At the strategic level managers aim to create a stable plan that remains fixed throughout the lifespan of a project [Turner 1993]. For the purposes of this thesis chapter one identified the metamodel as representative of a strategic level plan. As the metamodel incorporates any possible changes within its structure it provides a firm baseline from which to work. Any variation can be constrained to within the next level down - the tactical level. The tactical level is the detail at which most project managers tend to work. Most project management software tools are also aimed at this level of detail. At this level, specific tasks and responsibilities for those tasks, are identified. The software development example in chapter three represented a more detailed tactical level plan. It is at the tactical level that the project management functions identified above are targeted.

Breaking a project down through these levels, to ever increasing detail, represents the creation of a Work Breakdown Structure. This does not occur in one step, but through several levels of breakdown. Turner (1993) identified several advantages of using a Work Breakdown Structure:

- It provides better control of work definition
- It allows work to be delegated in coherent packages
- It allows work to be defined at an appropriate level for estimating and control of the current stage
- It allows risk to be contained within the Work Breakdown Structure

An example of a Work Breakdown Structure (for the Milltown Road Bridge project) can be found in Appendix A (page A1). This breakdown was generated with CA-SuperProject® (1993). The project was successfully completed in January, 1994.

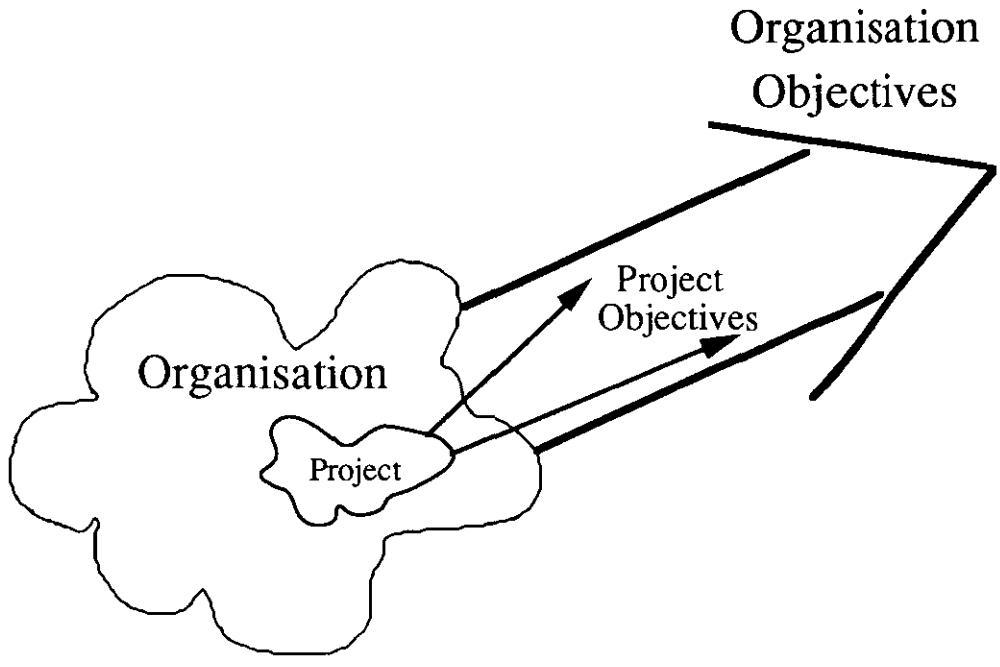


Figure 4.2 Project and Organisation Objectives

4.1.3 Milestones

By breaking a project down into a strategic level of detail (the metamodel level) each stage within a plan at this level represents a milestone. Milestones identify a measurable step on the way to completing a project. They provide a useful focus of attention for the project team and they highlight any problems in keeping to schedule, that in turn allows remedial action to be taken at the earliest opportunity. Milestones were identified in Generalised Activity Networks in chapter three by dashed nodes.

4.1.4 Activity Precedence

The precedence between activities at the tactical level is determined and represented as either an activity network (Activity-on-the-Arrow or Activity-on-the-Node) or as a linked Gantt chart. Linked Gantt charts are felt by some companies as the future of the project management industry. As an example, PowerProject® Version 2 (1991) by ASTA has no facility for generating Activity Networks and relies solely on a linked Gantt chart. It is unlikely that managers could just rely on linked Gantt charts as they tend to become rather cluttered with detail. Keeping the dependencies on a separate chart - an activity network - minimises this information overload.

4.1.5 Applying cost, resource, time, performance estimates to activities

At the tactical level, more accurate estimates of costs, durations and performance measures can be determined. In cases where these estimates are unclear, distribution functions are used (for example, Beta temporal functions in PERT) to provide a likely representation of durations or costs of activities. By using complex distribution functions, the analysis of these networks becomes more difficult.

Several approaches have been developed to determine costs and duration estimates of software development projects. Boehm (1981 and 1984) [DeMarco 1982] introduced the COCOMO (Constructive Cost Model) technique that determines the amount of effort and cost involved in developing a software product. Other metrics have also been devised in an attempt to quantify the development of software [Conte et alia 1986]. Activity temporal functions and their analysis are studied in more detail in sections 4.2 and 4.3.

4.1.6 Network Analysis

Activity networks provide more than just a visual representation of a potential project at a tactical level. By applying various parameters to activities within an activity network and performing some form of network analysis, estimates can be made of a project's likely cost, duration and performance. The temporal analysis of activity networks is studied in more detail in sections 4.2 and 4.3 where a method of analysis is improved and applied to Generalised Activity Networks.

4.1.7 Optimisation/Scheduling

Resources, such as people, finance, materials and machinery, are seldom available in unlimited quantities for use on any project [Goodman and Love 1980]. It becomes

necessary, therefore, to determine an optimum use of resources within a project. This is achieved by either employing various algorithms or trying a *what if* analysis.

Algorithms have long been used to smooth resource usage within projects and minimise costs and times with respect to one another and projects as a whole. Examples of algorithms developed for this purpose include Berman (1964), Burgess and Killebrew (1962), Clark (1961a), Levy et alia (1962), King (1964), Davis (1974), Woodworth and Willie (1975), Talbot (1982) and Brown (1988). Another way of scheduling a project is to select baseline dates. Turner (1993) identified three ways of selecting baseline dates for a project:

Schedule by early start - to motivate the workforce.
Schedule by late finish - presents progress in the best light.
Schedule in between - to smooth resource usage or determine the most likely outcome.

It is not necessary, however, to baseline a project within its initial time frame (early and late dates). Guerrieri (1987) identified two scheduling procedures that can be used to optimise either resource usage or times in a project:

(i) Resource levelling
Reduces the amount of variability in the pattern of resource usage for the duration of a project where there are sufficient resources available and the project must complete in a specified time.

(ii) Fixed Resources Limits Scheduling
Meets as close as possible the project completion date subject to fixed limits on resource availability.

Almost all project management tools mentioned within this thesis provide techniques for scheduling projects using mainly the approaches identified above. CS Project™ (1992), as another example, provides its own resource levelling approach called CARLO - Cost and Resource Levelling Optimisation. CARLO provides a scheduling algorithm that schedules activities with either the highest priority first, the least float first, or the least duration first. Other approaches to scheduling, noted as current research trends in this area, include the analysis of optimal float usage in projects, a study of the interaction between parallel and interconnected activities, and the calculation of resource constrained float.

What if analysis identifies the effects that various changes can have on a project. As an example, the Monte Carlo™ (1993) software package provides a form of *what if*

analysis. It allows a user to increase (default of 20%) or decrease (default of 15%) the duration of all activities globally within a project to see what effect this will have. The theory behind this is that if an initial activity in a project varies from its expected duration or cost by a particular amount, the chances are that estimates of other activities will be out by a similar factor. Other forms of *what if* analysis can identify the effects of activities overrunning their expected costs, what if a member of the project team leaves at a particular stage?, or what if the software is developed by department *X* in this order? and so on. By identifying possible problems in a project beforehand, contingency plans can be arranged because the results of various changes will have already been anticipated. *What if* analysis is also used to determine optimum project configurations, for example, with respect to costs, location, staffing sizes and so on. Artificial intelligent systems often include *what if* analysis to determine optimum project solutions. For example, Kunz et alia (1986) detail an artificial intelligent management tool that allows a manager to view several project alternatives concurrently and see what affects various changes have on them.

By improving network analysis in stage four of the management process detailed above, it allows more efficient analysis within this scheduling stage. *What if* analysis becomes more accurate and more quickly performed, and simulation of changes in levelling and time/cost optimisation can be improved.

4.1.8 Control

Control perhaps represents more of an activity performed by management during the synthesis of a software development. It involves capturing 'actuals' as a project progresses, comparing these actuals with the baseline plan and updating a project plan accordingly. Using metamodels for software development has already been shown to improve the visibility and controllability of software development projects by providing more stable baselines at a strategic level. Another technique that improves management control of projects is Cost Schedule Planning and Control or CSPC [Saitow 1969]. CSPC integrates cost and schedule data, it provides a concise picture of project progress, it allows several projects to be controlled simultaneously, and it reduces subjective estimation.

Another technique that improves control is the Department of Defence methodology - DOD 7000.2 called C/SCSC (Cost/Schedule Control Systems Criteria or CSPEC) [DOD 1975]. This methodology works on the principle of a *cost control cube* where three structures, a Work Breakdown Structure, a Cost Breakdown Structure, and an Organisation Breakdown Structure are combined. When controlling a project, this methodology works on the *earned value* principle. This identifies how much earned

value an activity has achieved and is defined in one of four ways:

(i) The percentage of progress. This is rather subjective and can suffer from the 90% syndrome mentioned below.
(ii) 0/100 measure. This only counts activities in the control process if they have been successfully completed.
(iii) 100/0 measure. This accumulates all earnings from an activity as soon as that activity is started.
(iv) 50/50 measure. This splits the earned value into a start and an end. Half the earned value would be achieved when an activity starts, and the remainder when it ends.

Project management software packages that employ this technique are InstaPlan™ (1990), Cascade® (1993), Parade® (1993) and CA-SuperProject® (1993). Parade® takes plans generated by Primavera Project Planner® (1994) and provides an earned value control analysis for them. Cascade®, that provides the user with a complete planning methodology, is covered in more detail below.

Trying to estimate how much of a task is complete as a project progresses is fraught with problems. This subjective estimate often suffers from the 90%/10% rule (or 90% syndrome) [Abdel-Hamid 1988]. This rule shows that people usually feel, after spending 10% of the expected time on an activity, that 90% of that activity is complete. It is usually the case, however, that the remaining 10% of work takes up the remaining 90% of the allotted time.

4.1.9 Larger Projects

The techniques outlined above relate to the more traditional idea of project management - that of planning and control from a strategic and tactical level. Also identified, however, was the need for project-based management that embodied all levels of an organisational structure. In large organisations, where projects can span several departments and constitute several mini projects, the functions, identified above, are performed at some level, somewhere along the line. Figure 4.1 shows a more modern, organisation-wide approach to project-based management. Cascade® by MANTIX Systems Limited (1993), used at both British Telecom at Martlesham Heath [Hurley 1993] and GPT at Nottingham, provides a methodology very similar to this structured approach. Cascade® provides an example of a large organisation-wide project management methodology based initially on the C/SCSC technique mentioned earlier. Managers using Cascade® begin by performing a Work Breakdown Structure, like the methodology above, but from a higher organisational level. The next stage in this

methodology is to determine an Organisation Breakdown Structure (OBS) that identifies particular work groups within the organisation. These work groups are assigned particular tasks, identified from the WBS, called Cost Accounts in a Responsibility Assignment Matrix (RAM). Within each Cost Account, in the RAM, budgets are assigned, costs and schedules are approved, and baselines are maintained. Responsibility for each Cost Account is assigned to a Cost Account Manager. In effect, the Cost Account Manager is responsible for their own mini-project that is approached in much the same way as the steps detailed earlier. Each mini-project is broken down, using a Work Breakdown Structure, into *work packages* that form a detailed schedule for a mini-project. A Cost Breakdown Structure (CBS) is then used to pull together all costs and resources required by the work packages. The CBS identifies how resources relate to one another, it pulls in information from rate tables and working calendars (that identify resource availability, overheads and so on), and it summarises costs by cost types. In these mini-projects dependencies are set up, enabling standard critical path analysis to be performed, and resources are levelled by spreading the resource requirement evenly over the duration of the work packages. The mini-projects are then monitored and controlled using the C/SCSC methodology described earlier. This information can be rolled up through each level, via any of the structures, to provide summary information to senior executives, departmental managers, program managers, financial managers, and project team members. Figure 4.3, adapted from the sales literature of Cascade®, provides a visual representation of this entire process.

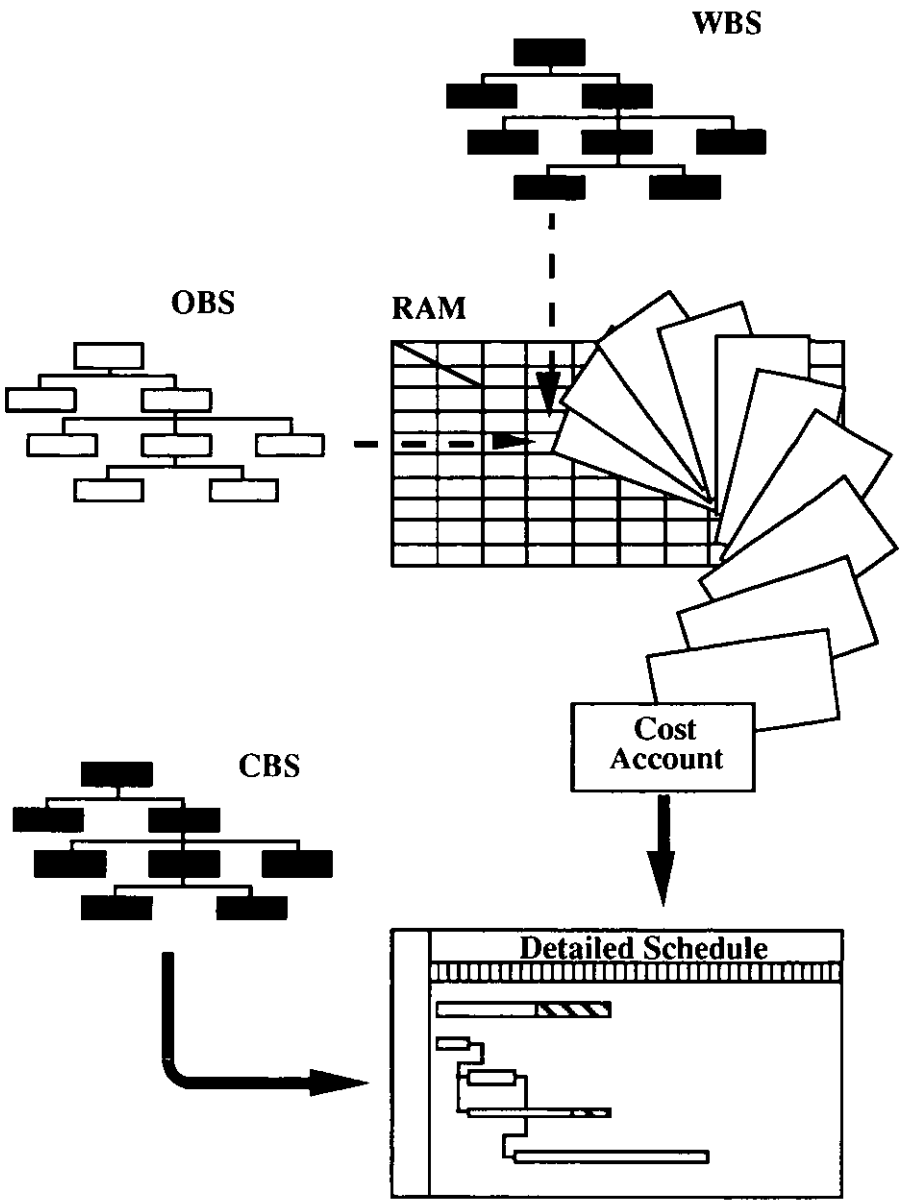


Figure 4.3 The Cascade® Methodology

4.2 TEMPORAL ANALYSIS OF ACTIVITY NETWORKS

4.2.1 Overview

Activity networks provide more than just a visual representation of a potential project. The analysis of activity networks at the tactical level, identified in stage five of the planning process, can often prove difficult. Temporal, cost and quality analysis is somewhat handicapped by the limited computational techniques available. Of these

three parameters, temporal analysis is the most difficult to perform due to the way in which activity temporal functions combine. Whereas costs combine in a rather simplistic additional form, activity durations can combine in minimum, maximum, or additional ways. For example, figure 4.4 represents a simple subnetwork (Activity-on-the-Arrow in this case) consisting of two activities A and B. The cost of performing this subnetwork is simply the additive costs of the two activities concerned. The duration of this subnetwork is more difficult to determine as it is calculated as the maximum duration of the two activity temporal functions. This is much more difficult to determine and is not based on simple additional rules. In the case when both activities are represented by Normal distribution functions, formulae from Clark (1961b) can be used. In most situations, however, simulation provides a more direct result.

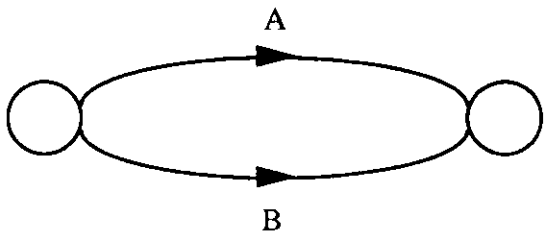


Figure 4.4 Simple Activity Network

More problems are introduced in the temporal analysis of Generalised Activity Networks. Figure 4.5 represents an example of a Generalised Activity-on-the-Arrow network that will complete, quite clearly, when all activities A, B and C, have finished. What is not clear from this network is which activity will be the last to finish. It is possible that activity B may take longer to perform than both A and C combined. The analysis of Generalised Activity Networks can prove to be particularly difficult as they are based on both activity temporal outcomes and the results of probabilistic logics. Simulation is seen as the most straight forward analysis approach to such networks.

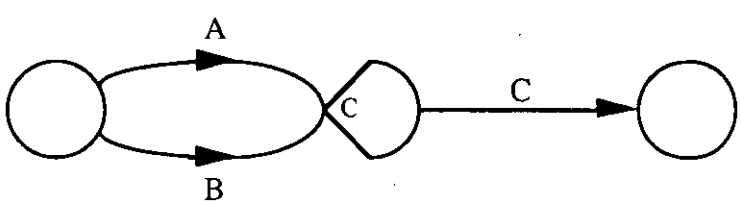


Figure 4.5 Simple Generalised Activity Network

An approach commonly used for the analysis of both Probabilistic and Generalised Activity Networks is Monte Carlo Simulation. Although this approach is perhaps the

most popular and 'accurate' technique available, it is not beyond improvement. How this technique can be improved, and how it is used in Generalised Activity Network analysis, is the subject of the latter part of this chapter. Monte Carlo Simulation is also used to assess the affects that various, known activity temporal functions have on an overall project duration. These results, along with various Probabilistic and Generalised Activity Network simulation results, are presented in the following chapter.

4.2.2 Deterministic Activity Networks

Due to the nature of activity temporal functions (ie constants) in Deterministic Activity Networks they are relatively easy to analyse. The original CPM technique [Kelley 1961, Moder and Phillips 1983, Levy et alia 1963] provided basic equations for making forward and backward pass calculations through Deterministic Activity Networks. For more detail of these equations, and calculations of various activity float times, refer to Goodman and Love (1980) and Appendix B.

4.2.3 Probabilistic Activity Networks

In Probabilistic Activity Networks the activity durations are represented by some form of distribution function (temporal function). Consequently, analysis of the completion time of these networks is particularly difficult and is sometimes referred to as the PERT problem. The classical PERT approach to this problem will be examined first, before highlighting its shortfalls and examining the alternative techniques.

4.2.4 Classical PERT

Probabilistic Activity Networks originated in the late 1950s when the original PERT methodology was developed by the United States Navy et alia (see chapter one) [Department of the Navy 1958]. The PERT methodology incorporates the understanding that the duration of activities within a project cannot necessarily be estimated accurately (as the CPM technique assumes). As a consequence, some form of subjective estimate needs to be made as to the duration of each activity within a project. An attempt was made to model the activity durations by some form of distribution function, from which various statistics could then be drawn. The PERT originators decided that a Beta distribution function (represented by $\beta(\alpha_1, \alpha_2)$) provided an acceptable representation as it can be manipulated into various shapes according to its two parameters, α_1 and α_2. Law and Kelton (1991) confirmed the original assumption that this Beta distribution function is probably skewed to the right (positively skewed) for real world activities. In this case $\alpha_2 > \alpha_1 > 1$ and the distribution resembles that shown in figure 4.5.

To generate an approximation of an activity's duration by a Beta distribution function, project managers are required to make three subjective estimates of an activity's duration: a, m, and b.

- a an optimistic estimate of the activity's duration
- m an estimate of the most likely duration of the activity (the mode)
- b a pessimistic estimate of the activity's duration

From these estimates the classical PERT approach provides two equations from which an estimate of the mean, t_e, and standard deviation, σ_e, of an activity's duration can be calculated - equations 4.1 and 4.2 respectively [Miller 1962, Goodman and Love 1980]. Figure 4.5 shows the relative positions of these estimates and the calculated mean, t_e, on a Beta distribution function.

$$t_e = \frac{a + 4m + b}{6} \qquad (4.1)$$

$$\sigma_e^2 = \frac{(b - a)^2}{36} \qquad (4.2)$$

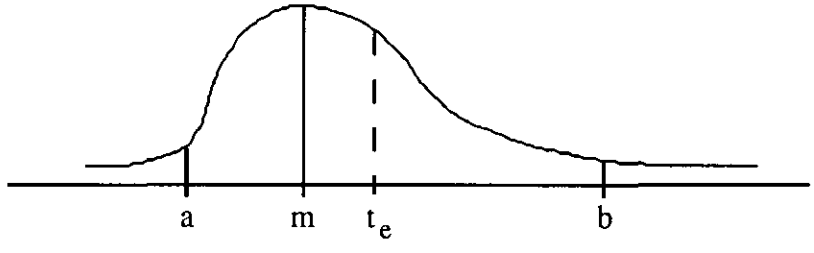


Figure 4.5 Beta Temporal Function

If one assumes, as the PERT originators, positive skew in this beta distribution function, the parameters of the Beta distribution, α_1 and α_2, can be calculated directly from the three estimates a, m, and b. Equations 4.3 and 4.4, adapted from Law and Kelton (1991), represent these calculations.

$$\alpha_1 = \frac{4m + b - 5a}{b - a} \qquad (4.3)$$

$$\alpha_2 = \frac{5b - a - 4m}{b - a} \qquad (4.4)$$

With the mean duration, t_e, and the standard deviation, σ_e, calculated for each activity in an activity network, it is possible to calculate the expected completion time of a project using the same approach as that used for Deterministic Activity Networks - forward and backward pass calculations (Appendix B).

Within a Probabilistic Activity Network there are K paths, P_j (j = 1, 2, . . . , K), each consisting of a number of activities i. Activity $i \in P_j$ if activity i lies on path P_j. Using equations 4.1 and 4.2 respectively, each activity, i, has associated with it an estimated mean duration, t_i, and standard deviation, σ_i. Thus, within the classical PERT approach, one can calculate the duration, D_j, of a path, P_j, in an activity network as:

$$D_j = \sum_{i \in P_j} t_i \qquad (4.5)$$

According to the classical PERT approach one can also calculate the variance, V_j, of this duration from:

$$V_j = \sum_{i \in P_j} \sigma_i^2 \qquad (4.6)$$

The classical PERT approach then dictates that the expected completion time of a project is based on the duration of the longest path through the network (the Critical Path). The variance of the Critical Path is also used to represent the variance of the project duration.

There are, however, several problems with the assumptions and calculations used in this classical PERT approach. As early as 1964 three problems were identified by MacCrimmon and Ryavec [MacCrimmon and Ryavec 1964]:

(i) The activity temporal function is not necessarily Beta distributed.
(ii) The mean and standard deviation calculations, 4.1 and 4.2, are incorrect.
(iii) Errors are introduced by poor estimates of the three parameters a, m and b.

Three other problems with the classical PERT approach can also be identified:

(iv) The assumption that there is one dominant, critical path is not always correct.
(v) For one-off projects the modal estimates of the project duration provide more realistic results than the mean.
(vi) The deterministic structure of the PERT network makes it difficult to model uncertainties that are inherent in many development projects.

(i) Problems with the Beta Distribution in PERT

It is unreasonable to assume that the duration of every activity within every project can be modelled by some form of Beta distribution function. MacCrimmon and Ryavec noted this problem in 1964 [MacCrimmon and Ryavec 1964]. They calculated the worst absolute error in the mean, $Err(t_e)$, and the worst absolute error in the standard deviation, $Err(\sigma_e)$, by incorrectly assuming a Beta distribution as:

$$Err(t_e) = \frac{1 - 2m}{3} \qquad (4.7)$$

$$Err(\sigma_e) = \frac{1}{6} \qquad (4.8)$$

Their assumption was based on the belief that an activity temporal function has at least the three properties of unimodality, continuity, and two non-negative abscissa intercepts. This, in fact, is not necessarily the case. Lootsma (1989) argued that discrete distribution functions would reflect activity durations more accurately. After all, an activity tends not to complete midway through a specific time unit. Lootsma stated that even though an activity could end during the middle of a day it would be unlikely that work would start on the next activity until at least the following morning. Activities usually finish in whole time units and this is modelled more realistically by discrete distribution functions. Pohl and Chapman (1987) also identified the need for discrete activity temporal functions and described two in their work - a 'bar' distribution and a 'spike' distribution. A 'bar' distribution represents an activity that can only be completed at specified time intervals (discrete) and a 'spike' distribution represents an activity that only completes at more disjointed discrete times (for example, an activity that relies on a committee that meets only four times per month). Pohl and Chapman also suggested other alternatives to the Beta distribution function including the Normal distribution, a Triangular distribution and a 'user determined' distribution. Various project management software packages also provide alternatives to the Beta temporal

function - for example, Predict!™ (1992) and @Risk™ (1990). The temporal functions these and other packages offer were detailed in chapter three.

Lootsma (1989) also suggested using the Gamma distribution function, because it is more natural than the original Beta distribution, and it is also simpler to analyse. Other authors have also used alternative activity temporal functions. Clark (1961b) used a Normal distribution function in his analysis, and Mongalo and Lee (1990) used Triangular, Rectangular, Beta and Normal distributions in their simulation analysis of PERT type networks. Lootsma (1989) also suggested using a Triangular membership function. Alternative distribution functions have also been used in the GERT technique [Moore and Clayton 1976]. This technique allowed up to ten possible distributions to be modelled for each activity duration (see section 3.5).

There is still much research needed into which distribution functions model best the duration of real world activities. Project managers are, on the whole, not statisticians and to expect them to estimate much more than three time estimates, as some do now with the classical PERT approach, is unreasonable. Pohl and Chapman (1987) noted, however, that 'A significant amount of time should be invested to accurately determine the probability function of each activity'. It is because time represents a significant cost to project managers that this investment is not achieved.

The affect that alternative activity temporal functions have on the overall project duration will be studied in detail in chapter five.

(ii) Erroneous Mean and Standard Deviation Estimates in PERT

The estimates of the mean and standard deviation of an activity duration (equations 4.1 and 4.2 respectively) are by no means precise. Indeed, MacCrimmon and Ryavec (1964) identified the worst absolute errors in these equations as:

$$Err(t_e) = \left| \frac{4m + 1}{6} - \frac{m(\alpha_1 + 1)}{\alpha_1 + 2m} \right| \qquad (4.9)$$

$$Err(\sigma_e) = \left| \frac{1}{6} - \sqrt{\frac{m^2 (\alpha_1 + 1)(\alpha_1 - \alpha_1 m + m)}{(\alpha_1 + 2m)^2 (\alpha_1 + 3m)}} \right| \qquad (4.10)$$

Improvements on these estimates have been explored over the years. Work by Golenko-Ginzburg in 1988 [Golenko-Ginzburg 1988] provided an improvement on these estimates - equations 4.11 and 4.12. Chae and Kim (1990) also suggested an improvement based on using the likelihood ratio of the mean and the midpoint and Donaldson (1964) provides an improved estimate of the variance based on one of the initial estimates being the mean.

$$t_e = \frac{2a + 9m + 2b}{13} \qquad (4.11)$$

$$\sigma_e^2 = \frac{(b - a)^2}{1268}\left(22 + 81\frac{(m - a)}{(b - a)} - 81\left(\frac{m - a}{b - a}\right)^2\right) \qquad (4.12)$$

Another suggested improvement to estimating an activity's mean duration comes from Whitehouse (1973). He suggested that the mean of the Beta distribution should be adopted as the third time estimate, m, rather than the mode. This would remove statistical inconsistencies inherent in previous values and allow for greater skew in the distribution. In this case the values of t_e and σ_e, derived from the Beta distribution, can be calculated from equations 4.13 and 4.14.

$$t_e = \frac{(\alpha_1 + 1)b + (\alpha_2 + 1)a}{\alpha_1 + \alpha_2 + 2} \qquad (4.13)$$

$$\sigma_e^2 = \frac{(b - a)^2 (\alpha_1 + 1)(\alpha_2 + 1)}{(\alpha_1 + \alpha_2 + 2)^2 (\alpha_1 + \alpha_2 + 3)} \qquad (4.14)$$

Lootsma (1989) argued the case for using the modal estimate of an activity's duration in a fuzzy model approach to the PERT problem. This makes more sense in one-off projects as the mode represents the most likely outcome of an activity in question. It also reduces the complexity of the calculations required when a network is analysed. The arguments for using the modal values for a network duration are discussed later.

(iii) Poor estimates of a, m and b in PERT

Not only are errors introduced by the mean and standard deviation estimates, as highlighted above, but errors are also introduced by possible erroneous estimates of a, m and b by a project manager. The worst absolute errors introduced by these estimates for the mean and standard deviation are:

$$Err(t_e) = \frac{a + 4m + 2b}{60(b - a)} \qquad (4.15)$$

$$Err(\sigma_e) = \frac{b + a}{30(b - a)} \qquad (4.16)$$

[MacCrimmon and Ryavec 1964].

Improving on these estimates rests on the shoulders of project managers themselves. It is only through experience that more accurate estimates of a, m and b can be made. How these figures are used, however, is the task of the theoretician.

(iv) The Single Critical Path Problem with PERT

The classical PERT approach highlighted the critical path as representative of an actual project duration. It has long been shown that this assumption can grossly underestimate project completion times [MacCrimmon and Ryavec 1964, Schonberger 1981, Anklesaria and Drezner 1986, Ballot 1989]. Although the critical path provides a useful focus of attention as a project unfolds, it should not be relied upon to provide an accurate estimate of a project's duration. Mongalo and Lee (1990) also highlighted this problem in their work when they compared PERT calculated network durations and Monte Carlo simulation results of various network types. Their work will be discussed in more detail later.

(v) Mode more accurate

In many cases it would make more sense to calculate the modal class of a network duration than the mean. This is certainly the case when network temporal functions are non-symmetrical and/or multimodal. In Generalised Activity Networks this is often the case. In symmetrical cases the mode and mean represent the same values anyway. For skewed and multimodal distributions this is not the case and the mode would certainly provide a better estimate of a project's duration, especially if the project was a one-off which most tend to be.

(vi) Inflexible Structure

The structure of a Probabilistic Activity Network, as was shown in chapter three, is purely deterministic in nature. By example in chapter three, a more generalised structure would allow real world activities to be mirrored more accurately. Any elaboration on this point has been covered in chapter three.

4.3 ANALYSING PROBABILISTIC ACTIVITY NETWORKS

Due to the inaccuracies introduced by the classical PERT approach, various techniques have being developed that attempt to solve, more precisely, the temporal analysis of probabilistic activity networks.

Many authors have categorised these approaches [Burt and Garman 1971a and 1971b, Robillard and Trahan 1977, Sculli and Wong 1985, Adlakha and Kulkarni 1989]. Adlakha and Kulkarni (1989) perhaps encompass these approaches more clearly than any other. The approaches can, broadly speaking, be split into six categories: Classical PERT (discussed above), exact analysis, approximation, bounds, miscellaneous approaches, and simulation.

Each of these approaches will now be discussed in turn before concentrating on Monte Carlo simulation in detail.

4.3.1 Analytical Solution

An exact duration of probabilistic activity networks can be defined in precise mathematical terms. Burt and Garman (1971a and 1971b), Fishman (1985), Adlakha and Arsham (1992), and Adlakha (1992) provide definitions of the solution, the main points of which are detailed here. It is perhaps worth noting that these definitions are independent of activity temporal functions within a network. This definition is based on an Activity-on-the-Arrow representation. The calculations for an Activity-on-the-Node network are equivalent.

A probabilistic activity network can be defined as a directed, acyclic network with a single source node, *s*, and a single sink node, *z*. A network consists of N arcs, each associated with an independent, nonnegative random variable, X_i (i = 1, 2, . . . , N), representing an activity duration. Each X_i has a known distribution function, f_i, on $[0,\infty)$. There are K, paths P_j (j=1, 2, . . . , K), through a network that link the source node, *s*, with the sink node, *z*. Activity $i \in P_j$ if activity i lies on path P_j. The duration of path P_j is a random variable, T_j, where:

$$T_j = \sum_{i \in P_j} X_i \qquad (4.17)$$

The problem of calculating the completion time of a probabilistic activity network can be defined as calculating the density function of the random variable:

$$D = \max_{j=1..K} T_j \qquad (4.18)$$

The distribution function of D is given by:

$$F_D(t) = \int_0^\infty \int_0^\infty \cdots \int_0^\infty h(t; x_1, x_2, \ldots, x_N) \prod_{i=1}^{N} f_i \, dx \qquad (4.19)$$

where $h(t; x_1, x_2, \ldots, x_N) = 1$ if $0 < \max_{j=1..K} \left(\sum_{i \in P_j} x_i \right) \leq t$

0 otherwise

This analytical solution is particularly complex and involves two stages of calculation. The first stage involves calculation of a joint distribution function of path times. The second stage involves the complex multivariate integration of 4.19. Since an exact computation of equation 4.19 is #P complete, according to Adlakha and Kulkarni (1989) there is no hope of developing an algorithm to solve this problem in a polynomial time. Thus, other techniques have been devised that avoid this work.

4.3.2 Simplification

The idea behind simplification is to streamline the problem into one that can be managed more easily by either computational or simulation techniques.

There are two ways in which one can simplify a Probabilistic Activity Network. One can either simplify the network as a whole, or simplify the activity temporal functions that constitute the network.

There are two ways to simplify a network as a whole:

(i) Approximate a network by one that has a high positive correlation with the original, yet is easier to analyse. This can be used in both simulation and analytical approaches. This is generally referred to as Control Variates [Burt et alia 1970, Burt and Garman 1971a]. This approach requires the construction of a similar network to that being analysed (with a high positive correlation). Although this may improve the speed of the calculations thereafter, the construction of this positively correlated network does take some time. In

simulation approaches to network analysis it is questionable as to whether control variates provide an adequate improvement in simulation overheads due to this construction time. Burt and Garman (1971b) use control variates to improve their simulation analysis of probabilistic activity networks.

(ii) Reduce a network by a series parallel reduction to a simpler form. Martin (1965) provides an algorithm for this approach. It is based, however, on the understanding that the activity temporal functions can be represented by simple polynomials. This introduces an alternative simplification technique - that of simplifying a network's activity temporal functions.

There are three ways in which Activity Temporal Function Simplification can be achieved:

(i) Assume discrete random variables. Based on this simplification, Hagstrom (1990) and Fulkerson (1962) provide algorithms for solving the PERT problem. These algorithms would certainly be of some use if it could be shown that discrete distribution functions provide a more realistic representation of activity duration, than continuous functions that are used at present. Their algorithms only work, however, if all activities within a network are of a discrete kind.
(ii) Manipulate the distribution parameters only. Clark (1961b) and Sculli (1983) used this approach when they assumed activity durations could be represented by Normal distribution functions. In this case, as the Normal distribution function is well understood, manipulation of the distribution parameters only is a logical approach.
(iii) Martin (1965) approximated activity temporal functions by simple polynomials. This enabled him to implement a series parallel reduction algorithm that reduced an entire network to a single activity with a known polynomial function. This approach is, perhaps, an over simplification of the problem as activity temporal functions can prove to be rather complex.

4.3.3 Bounds

An alternative approach to solving the PERT problem is not to try and determine an exact solution to the problem, but to determine boundaries on a network completion time. There are several approaches to bounding the expected duration of an activity network. Elmaghraby (1967) provides two estimates that bound the duration from below. In these cases the bounds represent optimistic estimates of a project's duration.

Lefèvere (1986) uses 'bounds for the expectation of linear combinations of order statistics' to determine upper bounds on the completion time of a project. These upper bounds represent pessimistic estimates. An upper bound was also determined by Kamburowski (1985a). In his calculations, although the exact forms of an activity's temporal functions do not have to be known, it was a requirement that their cumulative density function belonged within the NBUE class (although no explanation was provided for this).

Kamburowski (1985b) reviewed the various approaches to bounding the completion time of PERT type networks. He commented on, and compared, the approximations of Fulkerson (1962), Malcolm et alia (1959), McClellan (1969), Spelde (1977), Shogan (1977), and Kleindorfer (1969). Kamburowski then went on to provide algorithms, based on the series parallel reduction of Martin (1965), that reduced a Probabilistic Activity Network to a trivial form providing lower bounds on the expected completion time.

4.3.4 Miscellaneous Approaches

There are some alternative approaches to the PERT problem that cannot be put into any of the categories encountered above.

In 1989 Lootsma [Lootsma 1989] wrote a particularly interesting paper, using fuzzy numbers, that condemned all attempts to estimate accurately the expected completion time of Probabilistic Activity Networks. His argument was that more accurate approaches to the PERT problem produce non-tight plans that create conflicts between project leaders and subcontractors. Non-tight plans are those that involve float times that provide bargaining power to both project leaders and sub-contractors. Without such float (ie a tight plan) no conflict arises as there is no time to negotiate with.

In his work, Lootsma also used a Gamma distribution function within a classical PERT approach and modal values for a fuzzy approach. His argument was that, for one-off projects, modal values provide better estimates of a project duration than calculated means and standard deviations. The use of modal values has been discussed earlier and will be looked at in more detail in the simulation analysis of activity networks.

Another approach has been the use of Order of Processing diagrams by Fisher et alia (1985) but this never caught on as an analytical approach. Anklesaria and Drezner provided yet another solution in 1986 [Anklesaria and Drezner 1986] when they applied a multivariate approach to the PERT problem. They determined the duration of the r most critical paths in an activity network and consequently an estimate of its duration.

They compared their results with the classical PERT approach that they found to underestimate the completion time by as much as 30%. As they used the multivariate Normal distribution function for activity temporal function representation, their approach did not provide a general result.

4.4 SIMULATION

4.4.1 Overview

'A project network is an example of a problem that lends itself very well to simulation applications' [Badiru 1991].

The analytical approaches outlined above, according to Neumann (1984), require a great deal of computational effort, therefore simulation is widely used for evaluating Generalised Activity Networks and Probabilistic Activity Networks. It is worth noting, however, that although the analytical approaches can still prove costly in terms of computational time, they do provide valuable benchmarks for simulation techniques [Ragsdale 1989].

The general Monte Carlo technique for analysing activity networks involves sampling from each activity's time or cost distribution function and combining these samples to produce one simulation of the entire network. These simulations are combined to form a picture of how a network behaves 'on average'.

As a network is repeatedly simulated, various statistics are obtained that provide information of a network's characteristics. These data can be split into two groups: those that are automatically generated as a simulation proceeds, and those that require some manipulation to provide results. Information that is generated automatically includes raw network duration and cost values, path occurrence counts, activity occurrence counts, and critical path counts. These data are manipulated to calculate the mean network duration and cost, the variance and the modal classes for these parameters, criticality indices for paths and activities, completion probabilities for Generalised Activity Networks, and distribution and cumulative distribution functions for times and costs.

Several 'improvements' have been made to the general Monte Carlo simulation approach over the years. The main improvements to which are:

(i) Control Variates.

(ii) Antithetic Variables/Stratification (used later).
(iii) Cutsets [Sigal et alia 1979 and 1980].
(iv) Conditional Monte Carlo Simulation [Burt and Garman 1971a, Garman 1972, Dodin 1986].
(v) Quasirandom Points [Adlakha 1992].
(vi) Intelligent Simulation Methods.

4.4.2 Control Variates

Control variates [Burt et alia 1970] improve the technique by using localised analysis and simulation to simplify a network into a form that can be simulated more quickly with smaller sample sizes and fewer activity time generations. The application of this approach differs between Probabilistic and Generalised Activity Networks and, as a result, is avoided in the comparison of these two simulation approaches in chapter five.

4.4.3 Stratification

Stratification has also been referred to as the Latin Hypercube approach [@Risk™ 1990] and provides more efficient sampling than the Monte Carlo technique for symmetrical distribution functions. References to Latin Hypercube include McKay et alia (1979), Iman et alia (1980) and Startzman and Wattenbarger (1985). Stratification is actually a more refined version of the antithetic variable approach outlined in section 4.5.7 later.

4.4.4 Cutsets

Cutsets [Sigal et alia 1979] are used to reduce the simulation of an entire network to some subset of the original. However, determining a Uniformly Directed Cutset within a network can take some time. This must be compounded with the overall simulation time and, as such, reduces the effectiveness of this method. Within Generalised Activity Networks it is impossible to determine a Uniformly Directed Cutset due to the probabilistic nature of a network's structure. Uniformly Directed Cutsets are, therefore, avoided in the simulation techniques presented later.

4.4.5 Conditional Monte Carlo

Conditional Monte Carlo [Burt and Garman 1971a], that assumes independence of certain paths in a network, has also been used to improve the efficiency of simulation techniques. This technique was found to be ineffectual in most cases as there are seldom any independent paths in a network or many unique activities (those activities

common to all paths) to make this method worthwhile. Also the time to identify independent paths and unique activities must be compounded with the time to simulate a network, thus reducing the efficiency of this method.

4.4.6 Quasirandom Points

More recent developments involve using quasirandom points [Adlakha 1987] to improve the efficiency of activity network simulation. Quasirandom points were shown [Adlakha 1992] to be more efficient in simulating small activity networks (fewer than twenty nodes) than an antithetic variable approach. Ideally one could pick and choose which simulation technique to use depending on a network's type and size. In practice, however, one cannot predetermine the network to be analysed and one tends to stick with a more popular simulation approach for analysing all networks. The antithetic variable approach presented later represents a more popular approach.

4.4.7 Intelligent Simulation

The term 'intelligent' in this title is somewhat of a misnomer. It tends to imply some form of highly sophisticated intellectual reasoning within the simulation technique. The intelligent simulation approaches proffered by Cook and Jennings (1979) are by no means sophisticated. Cook and Jennings provide three improvements to the crude Monte Carlo Simulation approach - *min-max*, *path deletion* and *dynamic shut-off*.

min-max was originally suggested by Van Slyke (1963). It involves removing paths from a simulation based on the following criteria: The optimistic duration of each activity is used to determine the optimum critical path. The pessimistic time is then assumed and the pessimistic critical path determined. Any paths with a pessimistic duration less than the optimistic critical path time are eliminated from any future simulations of the network. Of course, for this to work, there must be some unique activities on these paths for any arc deletions to occur. If this were not the case, the entire network would require simulating. It is also rather unlikely that many paths would have a pessimistic duration less than the optimistic critical path length. It is therefore felt that this is a rather weak approach.

Path deletion After one hundred iterations of a network, any activities on paths that have not been flagged as critical are eliminated from future simulations. This again requires the presence of unique activities on these particular paths to be of any practical use.

Dynamic Shut-off This approach uses crude simulation to produce iterations from a network. If, after one hundred simulations, using a Kolmogorov-Smirnov test at the 0.05 level, there is no change in the cumulative density function of a network's duration, the simulation terminates. This is an extremely poor technique as the Kolmogorov-Smirnov test is a particularly weak test. Experience showed this test to be far too insensitive to changes in the density functions.

One other intelligent approach, proffered by Van Slyke (1963), is to limit the samples taken from arcs with a low criticality index to every K^{th} simulation (K>1). This can improve the efficiency of the simulation but can be difficult to implement for particular network types. If only the mean and variance of the project duration are required, Van Slyke also suggested fixing the duration of activities that are always critical to their calculated mean and variance. Once again, like the previous technique, the network requires some initial simulation to determine which arcs to fix.

4.5 ANALYSIS OF GENERALISED ACTIVITY NETWORKS

4.5.1 Why Simulation is Used for Generalised Activity Network Analysis

All the analysis looked at so far has been devoted to the temporal analysis of Probabilistic Activity Networks. The analysis of Generalised Activity Networks has been largely overlooked due to the scarcity of this method. Generalised Activity Network analysis falls into three categories - Flowgraph Theory, Monte Carlo Simulation, and Mathematical Analysis.

Flowgraph Theory can only be used when a Generalised Activity Network consists solely of Exclusive-Or input and output nodes. These networks are particularly rare. Appendix C provides the detail of this approach.

Usually, Generalised Activity Networks consist of many different connection types. Their analysis is therefore limited to either simulation or some form of analytical technique. When one tries to analyse a Generalised Activity Network mathematically one soon begins to realise the complexity of this problem. The complex calculations involved in analysing ordinary Probabilistic Activity Networks are compounded with the probabilistic branching (both independent, dependent and exclusive) that can be commonplace within a Generalised Activity Network. Not only this, but certain node input characteristics imply that network durations can be dependent on minimum (as

well as maximum) completion times of several independent temporal functions. Providing activity temporal functions can be approximated, in this case by Normal distribution functions, it is possible to determine the completion time of a network. Formulae for calculating these values is presented in Dawson (1994c) and covered in Appendix D.

The complex nature of cross connections within Generalised Activity Networks cannot easily be solved by mathematical means. This is why simulation proves to be the most practical approach to Generalised Activity Network temporal analysis.

To see how effective Generalised Activity Network simulation analysis can be, a comparison is made between the simulation requirements of both Generalised and Probabilistic Activity Networks in the next chapter.

4.5.2 Monte Carlo Simulation of Activity Networks

In the following sections the analysis of both Generalised and Probabilistic Activity Networks by simulation is studied in more detail. A Dynamic Sampling Technique is introduced that improves the efficiency of such simulations. Antithetic Variables are introduced and can be used in simulations, where possible, to improve the efficiency of the technique still further.

Identifying the number of simulations required in the analysis of activity network is an area that has not yet been addressed by the project management industry. Three packages that perform simulation in the analysis of activity networks are Risnet™ (1993), @Risk™ (1990) and Monte Carlo™ (1993). When personnel who use these products were interviewed, there was no clear way in which the number of simulations were determined. Generally a 'suck it and see' type approach was used. The packages provide users with initial default simulation values from which they would start to work (usually set at around 500). If this default value provides too much accuracy, or takes too long, the value can be lowered. If not enough accuracy is obtained the value can be raised. The Dynamic Sampling Technique introduced below, moves away from this approach by determining the number of simulations required to provide a desired accuracy in the results obtained. This is a more responsive solution to the problem than that adopted by these software packages.

4.5.3 The Dynamic Sampling Technique

This section is based on the work presented in Dawson and Dawson (1993b and 1994b).

When a network is simulated usually an attempt is being made to obtain, as quickly as possible, an 'accurate' estimate of particular network characteristics - usually the mean duration, μ, the variance, σ^2, and a representative temporal function. One way of achieving this is to ensure that a network is not simulated any more times than necessary to reach the required level of accuracy. The Dynamic Sampling Technique discussed below ensures this criteria is accomplished.

Let D be a random variable corresponding to a network duration and let $D^{(i)}$, i = 1, 2, . . , N, be N independent samples taken from the distribution D. Since there are thousands of samples, the Central Limit Theory states that:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} D^{(i)} \qquad (4.20)$$

is an unbiased and consistent estimator of μ, the true mean of D, since $E[\hat{\mu}] = \mu$ (where $E[\hat{\mu}]$ is the expectation of $\hat{\mu}$).

Also, s, from:

$$s^2 = \frac{1}{N - 1} \sum_{i=1}^{N} (D^{(i)} - \hat{\mu})^2 \qquad (4.21)$$

is an unbiased estimator of σ^2, the true variance of D. After some more manipulations:

$$\mathrm{Var}(\hat{\mu}) = \mathrm{Var}\left(\frac{1}{N} \sum_{i=1}^{N} D^{(i)}\right) = \frac{\sigma^2}{N} \qquad (4.22)$$

(because at this stage one can assume that the $D^{(i)}$s are independent) which implies that the larger the sample size N, the closer $\hat{\mu}$ should be to the true mean, μ.

The question arises as how N is chosen such that a 'good' estimate of μ and other network characteristics are obtained? A particularly large value of N could be used so that there is extreme confidence in the results, but this would be inefficient if a smaller sample size would have sufficed.

The basis of the Dynamic Sampling Technique is to simulate the network *n* times (usually *n* is set to one hundred), and calculate the required sample size (called $\hat{N}$) from

the results obtained for $\hat{\mu}$ and s^2. This process is repeated, refining the estimates of μ, σ^2, and hence $\hat{N}$, until $N \geq \hat{N}$. At that stage enough samples have been taken from the network to imply the confidence in the estimate of the true mean of the network duration is within the required limits. One can also work out a confidence interval for the variance of the network duration.

Say, for example, in the network under consideration the estimate $\hat{\mu}$, with a probability of 95%, needs to be within 1% of the true mean, μ

$$P[.99\mu \leq \hat{\mu} \leq 1.01\mu] = 0.95 \qquad (4.23)$$

This can be expanded to:

$$P\left[\frac{.99\mu - \mu}{s\sqrt{\hat{N}}} \leq \frac{\hat{\mu} - \mu}{s\sqrt{\hat{N}}} \leq \frac{1.01\mu - \mu}{s\sqrt{\hat{N}}}\right] = 0.95 \qquad (4.24)$$

where $\frac{\hat{\mu} - \mu}{s\sqrt{\hat{N}}}$ represents the Student t-distribution with $\hat{N}$ - 1 degrees of freedom.

For large values of $\hat{N}$ this corresponds to the Standard Normal Distribution[1].

Thus:

$$\frac{1.01\mu - \mu}{s\sqrt{\hat{N}}} = 1.96 \qquad (4.25)$$

from the tables of the Standard Normal Distribution.

It is therefore possible to estimate the required sample size, $\hat{N}$, from:

$$\hat{N} = \frac{1.96^2 s^2}{(0.01)^2 \mu^2} \qquad (4.26)$$

[1] For a value of $\hat{N}$=100 this assumption would be in error by at most 4% for the 95% confidence interval. For larger values of $\hat{N}$, such as those used later ($\hat{N}$>500), this error becomes insignificant.

where μ is estimated by $\hat{\mu}$, and s^2 represents an estimate of the true variance σ^2.

These results are based on the confidence in the estimates $\hat{\mu}$ and s^2 which are based in turn on, what are believed to be, independent random samples $D^{(i)}$ (i=1, 2, . . . , N) from a network. It is assumed that there is no correlation between the $D^{(i)}$s - in other words $\rho_j = 0$ for $j = 0, 1, 2, \ldots, N-1$ (where ρ_j is the correlation between $D^{(i)}$ and $D^{(i+j)}$).

4.5.4 Improving the Estimate of $\hat{N}$

It has been shown [Law and Kelton 1991] that simulation output data are usually correlated and so the estimates of μ and σ^2 may be biased in some way. Any correlation would not affect the sample mean, $\hat{\mu}$, that remains an unbiased estimate of the true mean. However, the estimate of σ^2 is affected by the correlation and it was shown by Anderson (1971) that:

$$E[s^2] = \sigma^2\left[1 - \frac{2\sum_{j=1}^{N-1}(1-\frac{j}{N})\rho_j}{N-1}\right] \qquad (4.27)$$

In practice there is a positive correlation between samples: $\rho_j > 0$ that results in $E[s^2] < \sigma^2$. The estimate could therefore underestimate the true variance to some degree and, as a consequence, could grossly underestimate $\hat{N}$ which in turn affects the accuracy of the simulated results. To avoid this underestimation of σ^2 the χ^2 (Chi-square) distribution function is used:

$$\chi^2 = \frac{(N-1)s^2}{\sigma^2} \qquad (4.28)$$

The upper bound (χ^2_u) of the 95% confidence interval for χ^2 can be estimated by:

$$\chi^2 = \frac{(\sqrt{2N-3} - 1.96)^2}{2} \qquad (4.29) \quad \text{(adapted from Conover (1971))}$$

Therefore if σ^2_u represents the upper bound for σ^2 at the 95% level, from equations (4.28) and (4.29):

$$\sigma^2_u = \frac{2(N-1)s^2}{(\sqrt{2N-3} - 1.96)^2} \qquad (4.30)$$

As σ^2_u is the upper bound of the 95% confidence interval for σ^2:

$$P[\sigma^2_u \geq \sigma^2] = 0.975 \qquad (4.31)$$

Thus, there is 97.5% confidence that this represents an upper estimate on the value of σ^2.

The following are typical values of σ^2_u for example values of N:

$N = 100$	$\sigma^2_u = 1.36s^2$
$N = 1000$	$\sigma^2_u = 1.09s^2$
$N = 2000$	$\sigma^2_u = 1.07s^2$
$N = 10^6$	$\sigma^2_u = 1.003s^2$
$N = \infty$	$\sigma^2_u = s^2 = \sigma^2$

Equation (4.26) is therefore adapted to:

$$\hat{N} = \frac{1.96^2 \sigma_u^2}{(0.01)^2 \hat{\mu}^2} \qquad (4.32)$$

The Dynamic Sampling Technique can now be implemented as:

(i) $N \leftarrow 0$
(ii) Simulate the network *n* times (*n* usually set to one hundred)
(iii) $N \leftarrow N + n$
(iv) Calculate $\hat{\mu}$, s^2
(v) Calculate $\hat{N}$ from equation (4.29 and 4.30)
(vi) If $N<\hat{N}$ then go to step (ii)
(vii) Required accuracy has been reached

4.5.5 Reducing Sample Size

To reduce the number of samples needed for a simulation still further, one can either reduce the confidence in the results or demand less accuracy from the mean duration estimate. For example, reducing the confidence in the result from 95% to 90% results in a reduction of 30% in the required sample size. A greater reduction can be achieved

by demanding less accuracy in the estimate of the mean. Reducing accuracy from within 1%, to within 2% of the true mean, results in a 75% reduction in sample size - clearly this is something worth considering should simulation time be an overriding factor.

4.5.6 Problems

For Generalised Activity Networks the temporal function of a project is not necessarily distributed Normally, and can prove to be distributed in any multimodal form. In these situations the variance of a network's temporal function proves to be extremely high and consequently affects the efficiency of the Dynamic Sampling Technique. Applying the Dynamic Sampling Technique in these particular cases is ill conceived as the results in the next chapter will show. It is also worth noting the problem of over setting n. For example, if, after 100 simulations, it is identified that 101 are required, the technique will perform 200 before finishing. This can be avoided by using smaller values of n but the additional computations involved compound with the simulation time. Usually, with sample sizes determined over 1000, $n = 100$ is a reasonable starting point.

4.5.7 Antithetic Variables

Another way of improving the Monte Carlo simulation technique is to make use of antithetic variables. Sullivan et alia (1982) highlight the efficiency of using such methods in the simulation of Probabilistic Activity Networks.

Antithetic variables induce a negative correlation between pairs of samples taken from each activity's time distribution function. The following explanation of antithetic variables is adapted from the work of Burt, Gaver and Perlas (1970).

Antithetic variables can be used in the simulation of activity networks in which the activity time and cost distribution functions are independent of one another. Figure 4.6 represents a single activity network for which T_{12} is a random variable corresponding to the duration of activity a_{12}, and D is the random variable corresponding to the network duration. Clearly $D = T_{12}$ for this simple case.

In order to simulate this network it is necessary to take samples from the activity's time distribution function and use these samples to build up a picture of the network temporal characteristics. One way of taking samples from the activity's time distribution, T_{12}, is to generate uniform random numbers from $R^{(i)} = [0,1]$ and transform these into realisations of T_{12} by $T_{12}^{(i)} = F^{-1}(R^{(i)})$ (where F is the distribution function

of T_{12}, and $T_{12}^{(i)}$ is the i^{th} sample from T_{12}).

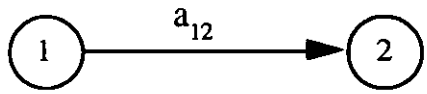


Figure 4.6 Single Activity Network

$D^{(i)} = T_{12}^{(i)}$ is the i^{th} iteration of this network. This process is repeated N times to provide an estimate, μ, of the true network duration mean, μ:

$$\hat{\mu} = \frac{T_{12}^{(1)} + T_{12}^{(2)} + \ldots + T_{12}^{(N)}}{N} \qquad (4.33)$$

The basis behind antithetic variables is, that in order to estimate μ, the samples $T_{12}^{(i)}$ (i = 1, 2, ... , N) need not be independent as long as they have the correct marginal distributions. In other words, if a sample from T_{12} is 'large' in one realisation it should be 'small' in another (and vice versa). This avoids skewing results excessively to either side of the mean. To achieve this, the sample $T_{12}^{(i)'} = F^{-1}(1-R)$ is generated at the same time that the sample $T_{12}^{(i)} = F^{-1}(R)$ is produced. $T_{12}^{(i)'}$ is the antithetic of $T_{12}^{(i)}$.

One can, therefore, calculate an alternative estimator of μ called $\hat{\mu}_A$ using sampled variables and their antithetics:

$$\hat{\mu}_A = \frac{T_{12}^{(1)} + T_{12}^{(1)'} + T_{12}^{(2)} + T_{12}^{(2)'} + \ldots\ldots + T_{12}^{(N)} + T_{12}^{(N)'}}{2N} \qquad (4.34)$$

$$\therefore \quad \hat{\mu}_A = \frac{1}{2}(\bar{T} + \bar{T}') \qquad (4.35)$$

where $\bar{T}$ is the mean of the sample, $T_{12}^{(i)}$, and $\bar{T}'$ is the mean of the sample $T_{12}^{(i)'}$ (i=1, 2, . . . , N).

By construction, $\bar{T}$ and $\bar{T}'$ are negatively correlated, ie $Cov(\bar{T}, \bar{T}') < 0$ (where Cov(X,Y) is the covariance between X and Y) and

$$Var(\hat{\mu}_A) = Var\left(\frac{1}{2}(\bar{T} + \bar{T}')\right) = \frac{1}{4}\left(Var(\bar{T}) + Var(\bar{T}')\right) + \frac{1}{2}Cov(\bar{T},\bar{T}') \qquad (4.36)$$

As $Var(\bar{T}) = Var(\bar{T}')$:

$$Var(\hat{\mu}_A) = \frac{Var(\bar{T})}{2} + \frac{1}{2}Cov(\bar{T},\bar{T}') < \frac{Var(\hat{\mu})}{2} \qquad (4.37)$$

Equation 4.37 shows that antithetic variables provide a more accurate estimate of μ than is achieved by doubling the sample size.

The most significant effect of antithetic variables is achieved when samples are taken from symmetric distributions. For example, assume that T_{12} is Normally distributed with mean μ_{T12} and variance σ_{T12}^2. A sample $T^{(i)}{}_{12}$ is generated from T_{12} by taking a value $r^{(i)}$ from the standard Normal distribution Nor[0,1]. $T^{(i)}{}_{12} = \mu_{T12} + r^{(i)}\sigma_{T12}^2$ is then calculated. The antithetic of $T^{(i)}{}_{12}$ can then be produced directly from:

$$T^{(i)'}{}_{12} = \mu_{T12} - r^{(i)}\sigma_{T12}^2 = 2\mu_{T12} - T^{(i)}{}_{12} \qquad (4.38)$$

which requires no further random number generation.

Using antithetic variables in activity network simulation provides three efficiency gains. First, only one random number needs to be generated to provide two samples from an activity's time distribution function. Second, as shown, they prove more efficient than doubling the sample size. Finally, in Generalised Activity Network simulation, not only are activities simulated but so is a network configuration that adds to the time of the simulation. Using antithetic variables allows two network simulations to be generated for each network configuration making the complete simulation more efficient.

4.6 MODAL CLASS DYNAMIC SAMPLING TECHNIQUE

The results in chapter five show how one cannot necessarily expect the temporal function of an activity network to be symmetrically distributed. For activity networks with multimodal or skewed temporal functions the mean and variance provide little information to a project manager. In these cases the mode (representing a most likely single time estimate) or the modal class(es) (representing a most likely range) provide more useful estimates of a project's completion time as they represent the most likely outcome of that project. As an example (taken from chapter five), figure 4.7 shows the duration histogram of an example Generalised Activity Network with low parallelism and low precedence (see chapter five). In this case the mean duration was calculated as 103 days with a variance of 110 days. The modal classes, in this case, provide a more

realistic estimate of this project's expected completion time. In this case the modal classes are 108 to 110 and 114 to 116 days. Usually one refers to one mode, or modal class, within a distribution. To refer to just one modal class in the example would ignore an equally possible project result. It is possible to have two or more intervals representing the most 'popular' project durations and all these intervals should be highlighted.

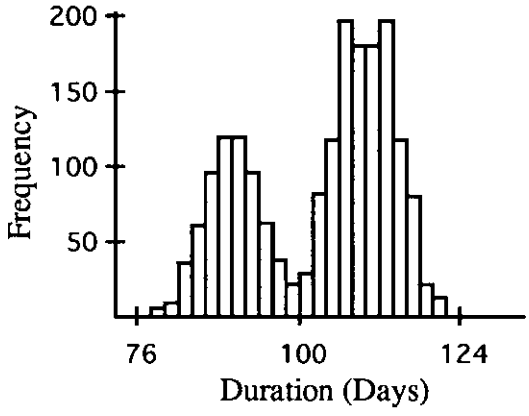


Figure 4.7 Example Generalised Activity Network Duration

Results of interviews with local software houses also identified a project manager's need for more realistic project outcomes such as those provided by modal classes rather than mean estimates. Managers prefer an indication of the most likely completion time of a one-off project rather than a mean value that represents the average completion time if a project was performed many thousands of times.

When the Dynamic Sampling Technique, introduced earlier, is used in skewed or multimodal cases, the variance tends to be so high that an unacceptable number of simulations are required to calculate an accurate estimate of the mean network duration. As this mean provides no real information, the Dynamic Sampling Technique requires some adaptation to deal with these situations. The alternative is a Dynamic Sampling Technique that attempts to home in, as quickly as possible, on the modal class of a network duration. An initial prototype algorithm is based on:

N represents the total number of samples from the network

(i) $N \leftarrow 0$

(ii) Simulate the network n times

(iii) $N \leftarrow N + n$

(iv) Calculate the Modal Class

(v) If there is a change in the Modal Class from the previous n samples go to step (ii)

(vi) The modal class has been established

Initial attempts to use this algorithm provide quite rapid results identifying a modal class rather quickly. There are, however, some shortfalls with this approach that require addressing before the full benefits of this technique can be achieved:

(i) There may be two or more similarly sized modal classes. At this stage this technique finds only one of them. Ideally the technique should identify the first r modal classes.

(ii) The technique appears to stabilise too quickly so one has reservations as to whether the true modal class has been established. The technique could, perhaps, be adapted to terminate only if a modal class has been identified three or more times in a row.

(iii) The range of the modal class cannot be predetermined. In this algorithm the range of the modal class is determined by the range of the network duration. Also, if one adjusts the size and number of class intervals the mode can jump around somewhat [Hays 1988]. One questions, in this instance, if the modal class determined by the technique is an accurate enough estimate.

(iv) It would be useful to determine the probability of a modal class occurring. Again, this is not achieved through this technique.

4.7 CHAPTER SUMMARY

4.7.1 Conclusion

This chapter began by introducing project-based management. Temporal analysis of activity networks is a sub part of this, assisting the management of time objectives at a

project's tactical and strategic level. The chapter went on to study approaches to the temporal analysis of activity networks and simulation was identified as the most popular approach to this form of analysis. A Dynamic Sampling Technique has been introduced to improve the efficiency of such methods. Antithetic variables have also been examined and shown to provide another improvement of simulation techniques. From this research, the need for a Modal Class simulation technique has been identified and its current shortfalls highlighted.

4.7.2 Future Work

From this work there are four areas that require further investigation.

First, the Modal Class Simulation of Activity Networks requires development.

Second, in real projects, if early activities over run their expected durations due to unforeseen circumstances and delays, the chances are that later activities will do likewise. This phenomenon is identified within the Monte Carlo™ (1993) software package that allows global changes of activity durations. It is likely that there is a positive correlation between delays in early and late activities. Introducing this reasoning into the simulation process would provide a more accurate model of real world processes.

Third, applying more representative distribution functions to activity durations needs researching. Discrete distributions may provide better estimates of activity completion times as activities tend to be completed in whole time units.

Finally, parallel algorithms can be used to reduce simulation costs still further (provided one has access to parallel processors). Initial inspection of this idea identifies two ways in which a parallel algorithm could be implemented:

(i) Simulate several activities concurrently.
(ii) Simulate several subnetworks concurrently.

It may also be possible to apply parallel algorithms to the complex problems of levelling and *what if* analysis. This could involve allowing various *what if* scenarios to be evaluated concurrently.

CHAPTER 5

Temporal Analysis of Activity Networks

CHAPTER PREFACE

Having outlined improvements to Monte Carlo simulation of activity networks (that are applicable to both Generalised and Probabilistic Activity Networks) this chapter presents, and discusses, the results of analysing these networks. Two results are presented in this chapter: First, the comparative simulation requirements of Generalised and Probabilistic activity networks from work based on Dawson and Dawson (1993b and 1994b), and second, the affect that various discrete and continuous activity temporal functions have on the duration of activity networks of different sizes and complexities (presented in Dawson (1994a and 1994b)).

CHAPTER KEYWORDS

Probabilistic Activity Networks, Generalised Activity Networks, Monte Carlo Simulation, Discrete Activity Temporal Functions, Continuous Activity Temporal Functions

5.1 INTRODUCTION

Three results are provided in this chapter - comparative simulation requirements of Probabilistic and Generalised Activity Networks, the affect of continuous activity temporal functions on the temporal function of a project, and the affect of discrete activity temporal functions on the duration of a project.

This chapter begins by detailing how the results of the network simulations are analysed both subjectively and objectively. It then moves on to present the results.

5.2 ANALYSIS OF RESULTS

5.2.1 Overview

When activity networks were simulated in this analysis, data values representing the duration of a planned project are generated. These data values were assessed by two different approaches - subjectively by graph plotting and objectively with the BestFit™ (1993) data analysis package. An example of the output data produced by BestFit™ is provided in Appendix E. In some cases a Normal Probability Plot was also generated to provide a subjective test for Normality. A representative subset of these results was then analysed using BestFit™ that applied three goodness-of-fit tests (see below) to determine which distribution function could best describe the data. BestFit™ was used to compare the durations of the activity networks with fourteen distribution functions - Poisson, Negative Binomial, Binomial, Geometric, Hypergeometric, Beta, Weibull, Triangular, Logistic, Chi-square, Normal, Lognormal, m-Erlang and Gamma. BestFit™ then ranked these functions in order according to which represented the data most accurately.

5.2.2 BestFit™

BestFit™ (1993) provides three goodness-of-fit tests (taken from Law and Kelton (1991) and outlined below) that are used to assess formally whether the data observations are independent samples from a particular distribution function, $\hat{F}$. BestFit™ matches the data to the most representative statistical distribution, testing the null hypothesis:

H_0: The data samples, X_i (i=1, . . , n), are independent and identically distributed random variables from $\hat{F}$.

(i) Chi-square Goodness-of-Fit Test (C-S Test)

The C-S test can be used to determine the best fit for both discrete and continuous distribution functions. Selecting the size and number of intervals, k, for the C-S test is particularly important as it can significantly affect the accuracy of results that are obtained. In most cases the number of intervals was defined implicitly by the range of discrete data obtained. In some cases, however, this range proved so large that a value of k was selected based on figures defined by Yarnold (1970). Yarnold stated that the C-S test would generally be acceptable provided that:

1. $k \geq 3$
2. $a \geq 5y(5)$

where $a = \min_{1 \le j \le k} np_j$,

y(5) is the number of np_j's less than 5,

and p_j is the proportion of data values in interval j.

For the data concerned, k was either assigned values ranging from fifteen and thirty to enable these criteria to be satisfied or was defined implicitly by the discrete output obtained.

The C-S test statistic (T-S), χ^2, is defined as:

$$\chi^2 = \sum_{j=1}^{k} \frac{(N_j - np_j)^2}{np_j} \qquad (5.1)$$

where N_j is the number of values in the j^{th} interval of the data. According to Law and Kelton (1991), the null hypothesis is not rejected providing that:

$$\chi^2 \le \chi^2_{k-1,1-a} \qquad (5.2)$$

For α at the 0.10 level, and for the values of k concerned, H_0 is not rejected provided that:

$$\chi^2 \le 21.064 \text{ for } k=15 \qquad (5.3)$$

$$\chi^2 \le 39.087 \text{ for } k=30 \qquad (5.4)$$

For other values of k (defined by the discrete data itself) the results were checked with required values from tables in Law and Kelton (1991).

(ii) Kolmogorov-Smirnov Test (K-S Test)

The K-S test can only be used to determine the best fit for continuous distribution functions. The K-S T-S, D_n, calculated by BestFit™ is defined as:

$$D_n = \sup_x \{ | F_n(x) - \hat{F}_n(x) | \} \qquad (5.5)$$

To compare the data samples with a hypothesised Normal distribution the mean and standard deviation of this distribution are estimated from the data. Having estimated

these values, D_n must be modified:

$$\left(\sqrt{n} - 0.01 + \frac{0.85}{\sqrt{n}}\right) D_n \qquad (5.6)$$

Thus, for a sample size of one thousand, the modified critical value for α at the 0.1 level means that H_0 is not rejected provided that:

$$D_n \leq 0.025885 \qquad (5.7)$$

(iii) Anderson-Darling Test (A-D Test)

The K-S test can only be used to determine the best fit for continuous distribution functions. The A-D T-S, A_n^2, calculated by BestFit™ is defined as:

$$A_n^2 = n \int_{-\infty}^{\infty} [\, F_n(x) - \hat{F}_n(x) \,]^2 \, \psi(x) \, \hat{f}(x) \, dx \qquad (5.8)$$

where $\psi(x) = \dfrac{1}{\hat{F}(x)[1 - \hat{F}(x)]}$

In order to compare the data samples with a hypothesised Normal distribution the mean and standard deviation of this distribution are estimated from the data. Having estimated these values, A_n^2 must be modified:

$$\left(1 + \frac{4}{n} - \frac{25}{n^2}\right) A_n^2 \qquad (5.9)$$

Thus, for a sample size of one thousand, the modified critical value for α at the 0.1 level means that H_0 is not rejected provided that:

$$A_n^2 \leq 0.6295 \qquad (5.10)$$

5.2.3 Tests for Normality

When activity networks are analysed one result of interest is the temporal function of the project they represent. Initial tests were based on ideas inspired by the results of

Mongalo and Lee (1990). They reported that the temporal function of an activity network was approximately Normal for all network types, even when the activity temporal functions were Triangular, Rectangular, Beta or Normally distributed. Based on this assumption a test for Normality is required that could handle up to five thousand simulations of an activity network. Ideally, one would hope that the duration of a probabilistic activity network could always be approximated by one of the more common distribution functions (such as the Normal) irrespective of which activity temporal function was used. If this proved to be the case, statistical results could be inferred directly from the characteristics of this known function. Simulation requirements could also be minimised as the function's parameters could easily be generated. Confidence intervals could also be obtained providing more accurate limits on project completion times and the Dynamic Sampling Technique could be used.

The tests available for Normality are:

(i) Chi-square Test

The Chi-square test is perhaps the most obvious choice as a test for Normality due to its popularity. However, Wetherill (1981) claims the Chi-square goodness of fit test is not a particularly good test for Normality in some cases. This is due mainly to the problem of combining cells in the tails of the distribution to avoid small frequencies. This is confirmed by the results presented later.

(ii) Normal Probability Plot

According to Wetherill (1981) the Normal Probability Plot provides a good, but subjective, approach to testing for Normality. With the number of samples dealt with (usually well over one thousand), although a Normal Probability Plot can take some time, it does provide a good visual indication of the Normality of a set of data. A Normal distribution would result in a straight (f(x)=x) plot on these axis.

(iii) W Test

Mongalo and Lee (1990) claimed to use this test based on the work of Royston (1982) and Shapiro and Wilk (1965). Unfortunately, this test is only applicable to sample sizes of less than fifty. This somewhat contradicts the results of Mongalo and Lee who claim to have used this test for 7500 simulation results. The results of simulating activity networks with different activity temporal functions also clearly contradict Mongalo and Lee's work (more on this later). The W Test was found to be of no practical use for sample sizes that are dealt with in these tests.

(iv) Calculation of Skewness and Kurtosis

For distributions suspected to be Normal, it is possible to calculate their skewness, γ_1,

and kurtosis, γ_2. Providing these values compare favourably with an equivalent, known Normal distribution function, one could perhaps claim Normality. To perform this test Wetherill (1981) provides the following test.

One carries out significance tests on γ_1 and γ_2, testing the hypotheses that $\gamma_1=0$ and $\gamma_2=0$ assuming that γ_1 and γ_2 are Normally distributed with expectation zero and calculable variances. For sample sizes greater than one thousand (which are required), the variances of γ_1 and γ_2 can be calculated as:

$$V(\gamma_1) = \frac{6}{n} \qquad (5.11)$$

$$V(\gamma_2) = \frac{24}{n} \qquad (5.12)$$

respectively (where n represents the sample size).

Based on a sample size of one thousand, and a 95% confidence interval on the coefficients, γ_1 and γ_2, one has

$$-0.005 < \gamma_1 < 0.005 \qquad (5.13)$$

$$-0.01 < \gamma_2 < 0.01 \qquad (5.14)$$

Provided these criteria are satisfied one could possibly assume Normality.

Unfortunately, when sampling directly from a known Normal distribution function, the values of γ_1 and γ_2 exceed these limits by quite some margin. For example, figure 5.1 shows the histogram and Normal Probability Plot of one thousand samples taken from a Normal distribution with mean 100 and variance 5. Although a test for Normality should prove positive in this case, the values of γ_1 and γ_2 from this data are:

$\gamma_1 = 0.02712$ and $\gamma_2 = -0.20926$

exceeding the calculated limits defined in 5.13 and 5.14. It was therefore felt that the γ_1 and γ_2 test for Normality was inadequate for samples of this size.

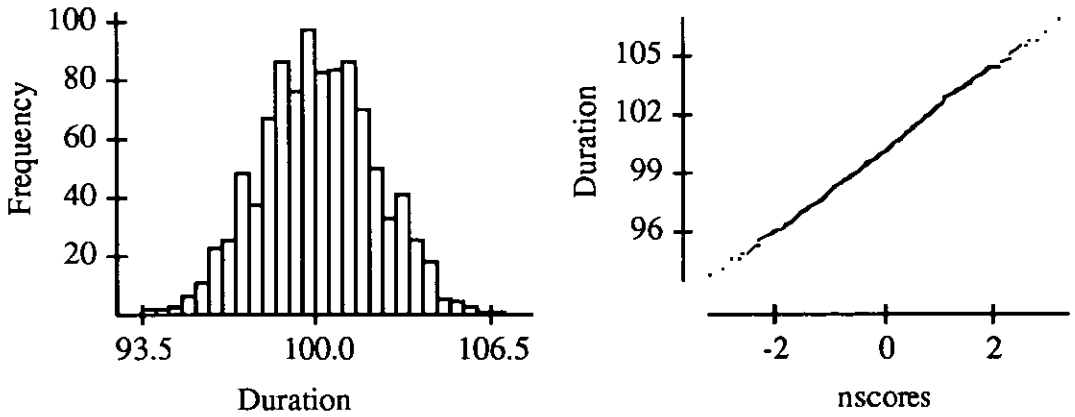


Figure 5.1 Histogram and Normal Probability Plot of Samples from a Normal Distribution

5.3 NETWORK CHARACTERISTICS

5.3.1 Overview

The configuration of an activity network significantly affects the work required and the accuracy of results that are obtained when that network is analysed. Three network characteristics, identified by Mongalo and Lee (1990), are size, precedence (sometimes called *fatness*), and parallelism. Table 5.1, which is adapted from Mongalo and Lee (1990), identifies a tighter quantification of these characteristics based on Activity-on-the-Arrow networks. The nodes and activity dependencies (the arrows) in Activity-on-the-Node networks do not lend themselves directly to these criteria. To determine the characteristics in Activity-on-the-Node networks either alternative definitions are required or they must first be converted to an equivalent Activity-on-the-Arrow representation.

Although the size measure is self evident, the two other measures require some clarification. The Criticality Index measures the probability that a given path through a network has a longer duration than any other path. The Critical Path is the path with the highest such probability. In Probabilistic Activity Networks the Criticality Index is the probability that a given path has a longer duration than any other path in the network. In pure Exclusive-Or Generalised Activity Networks this index is based solely on paths' probability values. In Generalised Activity Networks with several node characteristics this index is based on the duration of paths, on the probability that paths occur, and the probability that other paths do not occur. Calculating Criticality Indices based on maximum time characteristics (for example Clark (1961b)) and/or probabilities is

particularly difficult. Network simulation provides an alternative way of generating these results.

Precedence provides an indication of the complexity of a network structure based on the number of paths it contains. A network with high precedence would be fatter than one with low precedence.

In an Activity-on-the-Arrow network, given the number of activities, *Acts*, and the number of nodes, *Nodes*, it is possible to calculate the maximum, *Max*, and minimum, *Min*, number of possible paths through that network. Calculating the exact number of paths is impossible from these data alone and it is best left to simulation to provide such results.

Given that $Q = \text{Quotient}\left(\frac{\text{Acts}}{\text{Nodes} - 1}\right)$ and $M = \text{Modulus}\left(\frac{\text{Acts}}{\text{Nodes} - 1}\right)$

$$\text{Min} = \text{Acts} - \text{Nodes} + 2 \tag{5.15}$$

$$\text{Max} = (Q + 1)^{M} * Q^{(\text{Nodes} - 1 - M)} \tag{5.16}$$

Clearly, if $M = 0$, this reduces to:

$$\text{Max} = Q^{(\text{Nodes} - 1)} \tag{5.17}$$

The maximum number of paths in a network, and consequently its complexity, can prove to be quite considerable even for small networks. For example, with only three nodes and ten activities there can be up to twenty five possible paths through a network. Adding only one more activity can raise this number to thirty.

Size	Small	Medium	Large
Number of activities	under 26	26 to 80	over 80
Parallelism	Low		High
Criticality Index of the Critical Path	over 80%		under 80%
Precedence	Low		High
$\frac{\text{number of activities}}{\text{number of nodes}}$	under 1.5		over 1.5

Table 5.1 Network Characteristics

Mongalo and Lee's [Mongalo and Lee 1990] main interest was in the comparative accuracy of Monte Carlo simulation and standard PERT calculations based on a single critical path. They showed that whilst precedence had no affect on the comparative results obtained by both methods, both parallelism and size did have an affect. Mongalo and Lee used four activity temporal functions in their analysis - Triangular, Rectangular, Beta, and Normal. They used the W Test for Normality [Royston 1982] in all cases to test the distribution function of the network duration. In all variations of network characteristics this test for Normality did not fail. These results must be viewed with some caution as the W Test for Normality is a particularly weak test and cannot be applied to sample sizes with which they were working. Assuming Normality in all cases can also prove fallacious as the results below show.

In contrast, results presented later show that the duration of a Generalised Activity Network cannot be relied upon to assimilate a Normal distribution. When analysing the results of these networks a manager should be more interested in modal classes than the mean, and in the cumulative distribution functions of time and cost that provide probabilities of completing projects within particular limits.

5.3.2 Generalised Activity Networks

As it is of interest to compare simulation requirements of both Generalised and Probabilistic Activity Networks, the sample set of Generalised Activity Networks will be generalised somewhat (otherwise there would be an infinite sample set). For comparative purposes both Pure Exclusive-Or Generalised Activity Networks and Pure

Independent-Or Generalised Activity Networks will be compared.

As chapter three highlighted, Generalised Activity Networks allow loops to be formed within a network providing a useful approximation to real life projects where certain tasks are repeated. For the purposes of this analysis, loops have been avoided in the sample networks so that a clearer comparison can be made with Probabilistic Activity Network simulation in which loops are not allowed.

(i) Exclusive-Or Generalised Activity Network

This form of Generalised Activity Network allows only Exclusive-Or input and output logics at each network node. This implies that only a single path can be performed during one network simulation. Mathematically this network would appear to be the simplest to analyse. The mean of the network completion time, μ, and the variance of this time, σ^2, can be calculated from the simple pseudo code algorithm:

There are K paths, P_j (j=1, 2, . . . , K), in the network.
Let $P(P_j)$ be the probability of taking path P_j, and $T(P_j)$ the expected duration of path P_j.

for i = 1 to K do **begin**

$$T(P_j) = \sum_{i \in P_j} \text{mean duration of activity i}$$

$$P(P_j) = \prod_{i \in P_j} \text{probability of performing activity i}$$

end

then:

$$\mu = \sum_{j=1}^{K} P(P_j)\, T(P_j) \qquad \sigma^2 = \sum_{j=1}^{K} P(P_j) T(P_j)^2 - \mu^2$$

Unfortunately this algorithm can prove formidable because the number of paths is difficult to determine precisely in complex networks. It has already been shown how there can be a considerable number of paths even in small networks. For larger networks this algorithm proves to be wholly inappropriate. Simulation, therefore, provides a simpler alternative. The validity of using the mean and variance for analysing

the project completion time in Generalised Activity Networks must also be questioned as the results presented later show.

(ii) Independent-Or Generalised Activity Network
In this configuration node outputs were defined as Independent-Or and node inputs as W-Or (as defined in chapter three). This implied that nodes were realised when the last activity leading to them was completed or eliminated. In some cases, when all incoming activities are eliminated, nodes are not realised. If the sink node is not realised this implies that the network has 'failed'. This situation is unique to the Independent-Or Generalised Activity Network. In this case network simulation can be used, not only to estimate the project completion time, but also to estimate the expected probability of completing a project successfully.

The reader should be aware that it is possible to combine the activity characteristics from each network type to form an 'impure' activity network. Simulation analysis of an impure network is beyond the scope of this chapter since it presents an infinite sample set.

5.4 COMPARATIVE REQUIREMENT RESULTS

5.4.1 Overview

The first set of results presented are concerned with the comparative simulation requirements of Generalised and Probabilistic Activity Networks.

The comparative results provided two parameters that signified the simulation costs for each network type. The first value was a total count of the number of times activities in a network were sampled in each simulation. The second measure was a basic tick count (1/60th second) based on CPU time. These simulations were running on an Apple Macintosh™ LC 5/40. As one would expect the results showed that the ratio of the tick count to the number of activities sampled remained stable for each network type and configuration. However, this ratio differed quite significantly between Probabilistic and Generalised Activity Networks. The reason for this is that Generalised Activity Networks need to select which of an network's paths are simulated each time a network is iterated. This consequently compounds the simulation time. Probabilistic Activity Networks, on the other hand, need to sample every activity as all paths occur in each simulation. Consequently, the ratio of the tick count to activity samples proved to be higher in Generalised Activity Network simulation than in the comparative Probabilistic

Activity Network simulation.

Tick counts were chosen as the comparator between the two network types since they provide a more meaningful comparison of the network simulation costs than activity sample counts alone. As developments are made in Generalised Activity Network simulation analysis, it is anticipated that the time to generate a network configuration during each simulation will fall dramatically (for example, using control variates). It will then be more feasible to compare the network simulation requirements by activity sample counts alone.

Each configuration of size, precedence, and parallelism identified in table 5.1 was simulated for all network types using the Dynamic Sampling Technique detailed in the previous chapter. In all cases the required accuracy in the mean duration was set at 0.2% with 95% confidence and the simulation stepsize was set to one hundred. For the purpose of comparing Probabilistic and Generalised Activity Networks simulations, activities were represented by Normal temporal functions throughout. This allowed antithetic variables to be used, improving the efficiency of the simulations, and ensuring that all networks were assessed on the same footing. Two main results came out of these simulations.

5.4.2 Generalised Activity Network Temporal Function

Generalised Activity Networks cannot be relied upon to have a temporal function that is symmetrical and unimodal (ie a Normal type). Figure 5.2 shows a histogram of the duration of a Generalised Activity Network highlighting the possibilities. This figure represents the duration of a large Exclusive-Or Generalised Activity Network with high parallelism and high precedence. It is accompanied by the Normal Probability Plot. Figure 5.3 provides another example. In this case it represents the duration of a small Independent-Or Generalised Activity Network with low parallelism and low precedence.

When the duration of a Generalised Activity Network is multimodal, although the variance and mean duration can be calculated, the usefulness of these values must be questioned. In these situations the modal class(es) would provide a more sensible measure of a project's duration as it represents the most likely outcome of that project. In the example shown in figure 5.3 the actual mean completion time of this project was calculated as 103 days with a variance of 110 days. The modal classes, in this case, would provide a more realistic estimate of the project's completion time. In this case there are two modal classes - 108 to 110 and 114 to 116 days - both with equal probability. A manager should be more interested in these values than the mean duration. S/he would also be very interested in the secondary peaks around 88 days.

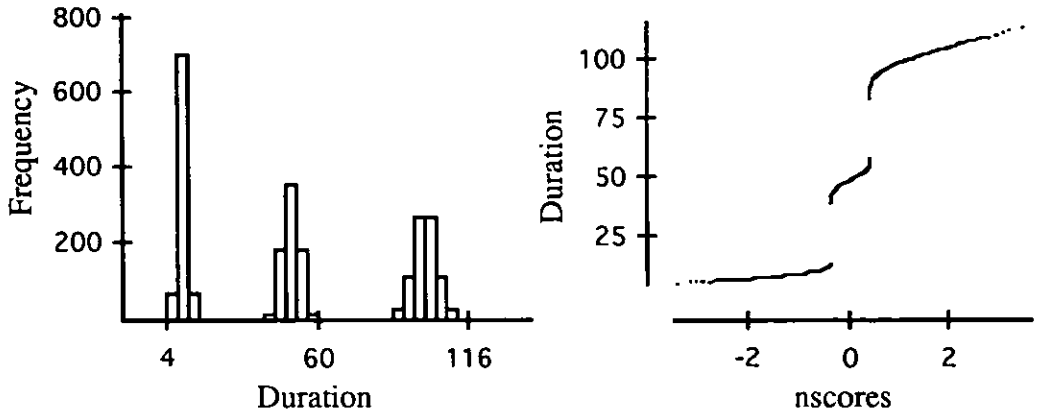


Figure 5.2 Example of a Generalised Activity Network Duration

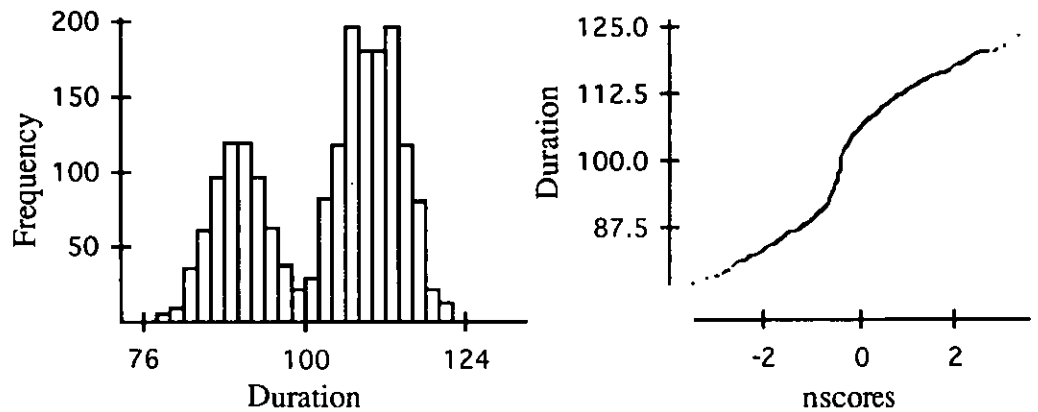


Figure 5.3 Example of a Generalised Activity Network Duration

Figure 5.4 provides a more useful representation of the duration of this project - a cumulative frequency distribution. Cumulative frequency distributions are often provided by project management software tools (for example, Risnet™ (1993), Monte Carlo™ (1993), and @Risk™ (1990)) to provide a more useful representation of project outcomes. They provide the probabilities of completing projects within particular time and cost limits. In this case there is only a 50% chance of completing this project within 107 days - 4 days longer than that indicated by the mean duration. Pohl and Chapman also identified the usefulness of cumulative density functions and claimed that they 'form the basis for project risk assessment or risk management'.

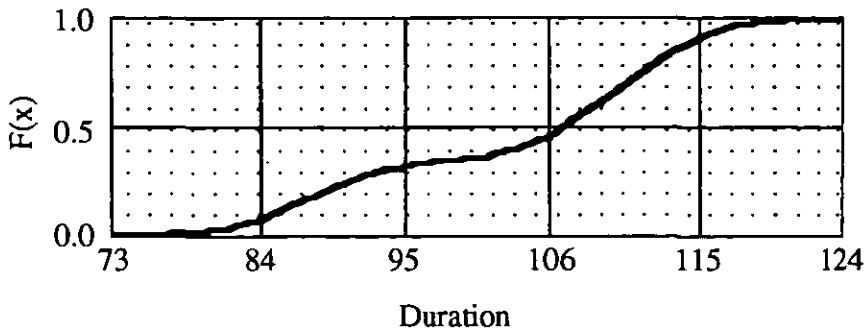


Figure 5.4 Cumulative Frequency of a Generalised Activity Network

The results for Probabilistic Activity Networks confirmed those of Mongalo and Lee - that the distribution function of a project's duration does approximate a Normal distribution in all cases (provided, at this stage, the activity temporal functions are Normal as well). Figure 5.5 shows a typical example of a Probabilistic Activity Network duration - in this case the network was large, had a low precedence, and low parallelism. It is accompanied by the Normal Probability Plot of this data.

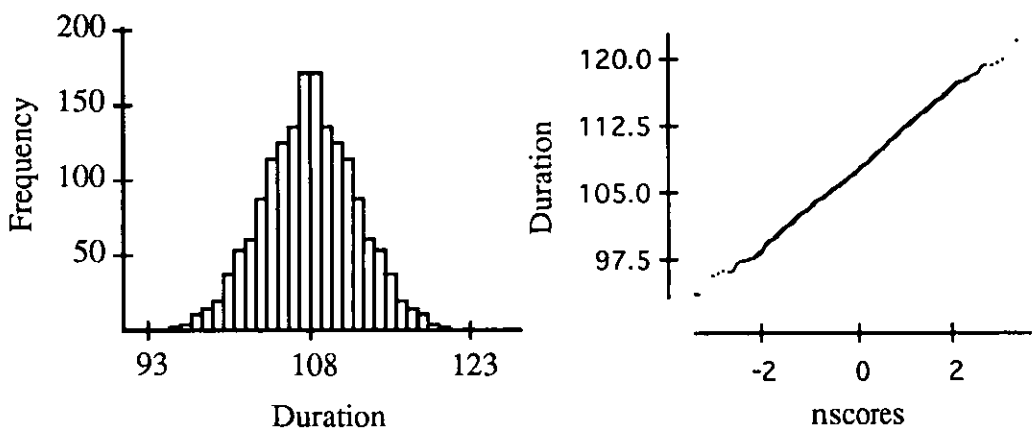


Figure 5.5 Example of a Probabilistic Activity Network Duration

The Dynamic Sampling Technique worked particularly well for Probabilistic Activity Networks but proved costly in terms of simulations required for non-Normally distributed Generalised Activity Networks. In these cases a high variance in the network duration implied the need for a large sample size to provide a reasonable estimate of the mean network duration. As shown above, the mean itself provides little information to a project manager. The technique, therefore, requires some adaptation to provide a reasonable estimate of the modal class(es) or the cumulative frequency distribution of a project's duration. The modal class simulation approach that was introduced in chapter four provides a more logical alternative.

5.4.3 Generalised Activity Network Normally Distributed

When the distribution function of the duration of a Generalised Activity Network is approximately Normally distributed (for example with a highly dominant critical path, or when the duration of dominant paths are similar) it takes at most the same time to simulate as the comparative Probabilistic Activity Network. This is particularly noticeable in networks with a high precedence. With low precedence and high parallelism the Generalised Activity Networks simulation is at worst within 20% of the equivalent Probabilistic Activity Network simulation time. Networks with low precedence are particularly uncommon and it is worth noting that these networks, due to their relative simplicity, are perhaps easier to analyse by alternative techniques (perhaps analytical). The results obtained show a clear improvement in the time to simulate Generalised Activity Networks for common high precedence networks. The reason for this is clear. Probabilistic Activity Networks require that all activities are sampled each time a network is simulated. A Generalised Activity Network on the other hand, although requiring a network's configuration to be generated, does not require that all activities be sampled each time. This proves to be a great time saving and results in a much faster simulation process.

The reader may have noticed that Probabilistic Activity Network and Generalised Activity Network simulations could be improved by only sampling from activities or paths which are dominantly critical - for example, incorporating the simulation approaches used by Van Slyke (1963). This chapter is concerned with the comparative simulation requirements of Probabilistic Activity Networks and Generalised Activity Networks and is not, therefore, concerned with streamlining the simulation techniques at this stage.

Table 5.2 shows the number of tick counts required for each network configuration. In these simulations Generalised Activity Network activity probabilities and time distribution functions were selected to ensure that these network durations approximated a Normal distribution. The starred (*) values were from distributions that were beginning to show signs of non-Normality. This lead to higher variances and consequently required larger than expected simulation times. Figure 5.6 represents the duration of the smaller starred network and figure 5.7 the medium (again they are accompanied by Normal Probability Plots) from table 5.2.

Size	Parallelism	Precedence	PAN	Ex-Or GAN	Ind-Or GAN
Large	High	High	8,180	3,123	5,263
Medium	High	High	5,526	2,354	3,115
Small	High	High	1,836	1,537	1,690
Large	High	Low	53,136	62,133	54,892
Medium	High	Low	32,690	35,857	35,050
Small	High	Low	7,066	7,864	7,393
Large	Low	High	46,485	2,271	5,066
Medium	Low	High	24,889	2,127	3,174
Small	Low	High	4,883	2,123	1,683
Large	Low	Low	53,123	79,412*	62,095
Medium	Low	Low	31,102	53,846*	33,842
Small	Low	Low	5,680	17,475*	7,154

Table 5.2 Tick Count Results of Comparative Simulations

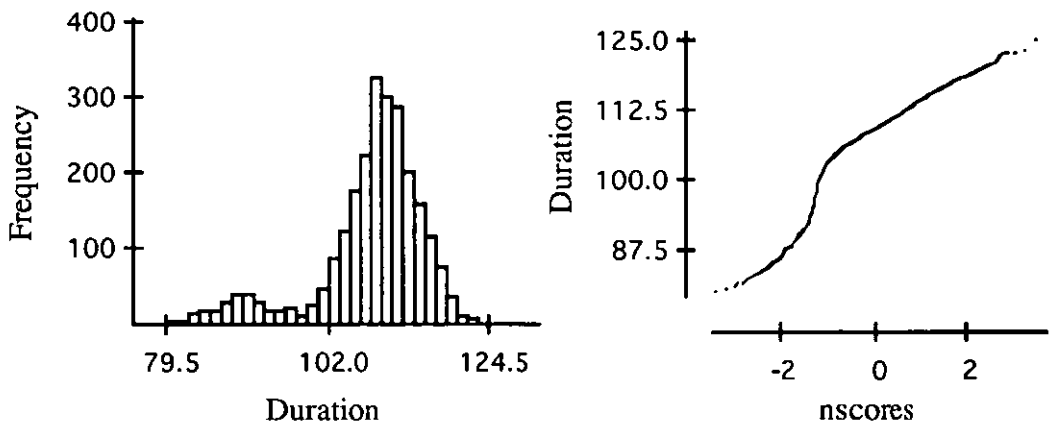


Figure 5.6 Small, Exclusive-Or Generalised Activity Network Duration

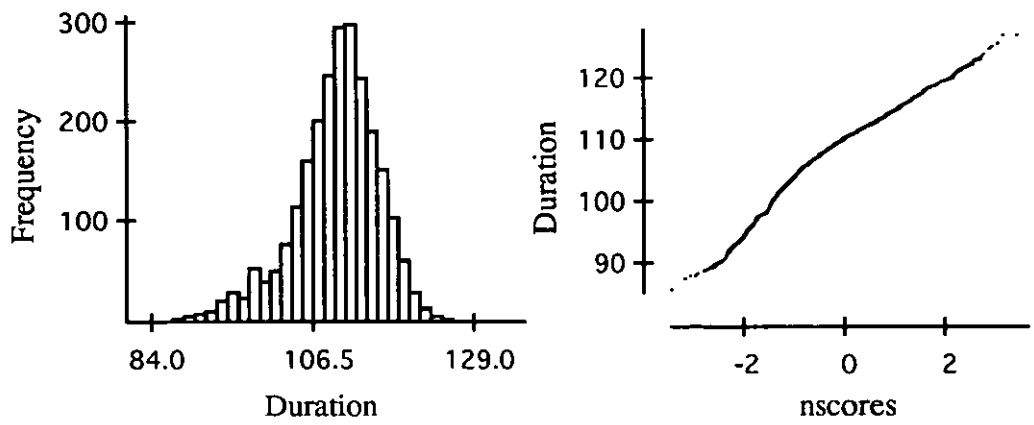


Figure 5.7 Medium, Exclusive-Or Generalised Activity Network Duration

5.4.4 Summary of Comparative Simulations

Simulation has proved to be a useful way of analysing both Probabilistic and Generalised Activity Networks. In Generalised Activity Networks however, more useful simulation results are those giving modal intervals and cumulative frequency distributions. The use of a Dynamic Sampling Technique and antithetic variables provided an improvement on a standard Monte Carlo simulation approach. The main results show that Generalised Activity Network duration distribution functions do not always assimilate a Normal distribution, but when they do, they prove to be quicker to simulate than the comparative Probabilistic Activity Network for common high precedence networks. The uncommon low precedence networks take similar times to analyse and may be better suited to an analytical approach than Monte Carlo simulation.

5.5 SIMULATION WITH KNOWN ACTIVITY TEMPORAL FUNCTIONS

Another area of analysis is the affect that different activity temporal functions have on an overall network duration. The motivation behind this work was the results presented by Mongalo and Lee (1990). It has already been shown that Generalised Activity Network's temporal functions cannot be relied upon to assimilate a Normal distribution function in all cases. The temporal function of Probabilistic Activity Networks, however, has so far been seen to be approximately Normally distributed. In this section each network configuration in Probabilistic Activity Networks was simulated with six different continuous activity temporal functions - Normal, Lognormal, Gamma, Beta, Rectangular, and Triangular and five discrete temporal functions - Binomial, Poisson, Negative Binomial, Bernoulli, and Discrete Uniform.

Each network configuration identified in table 5.1 was simulated one thousand times using these discrete and continuous activity temporal functions. Algorithms to generate the activity temporal functions were obtained from the following sources (for more detail refer to appendix F):

Bernoulli Distribution [Law and Kelton 1991, pp496-497] Used to represent two possible outcomes of an activity duration.

Binomial Distribution [Law and Kelton 1991, p502] Used to represent the likely successful completion of an activity in a discrete number of time units with probability p.

Discrete Uniform Distribution [Law and Kelton 1991, p497] Used when there is equal probability of an activity completing any discrete time during some finite period.

Negative Binomial Distribution [Law and Kelton 1991, p503] Used to provide a discrete function with identifiable skew.

Poisson Distribution (adapted from Ahrens and Dieter (1974)) Used to represent an activity with an estimated mean completion time in discrete units.

Normal Distribution [Rubinstein 1981, p90] The Normal distribution has been used by several authors (for example Clarke (1961b) and Mongalo and Lee (1990)) to represent the duration of activities in projects. Pohl and Chapman (1987) identified the Normal distribution function as that used for activities that have been performed several times before and in which the probability of unusual delays is very small.

Lognormal Distribution [Law and Kelton 1991, p492] Selected due to its ability to represent a skewed temporal function.

Rectangular Distribution (Generated by a simple algorithm) Some activities could have equiprobable continuous durations and are best represented by this distribution function (used by Mongalo and Lee (1990)).

Triangular Distribution (adapted from Law and Kelton (1991), p494) Used by Lootsma (1989) as a simpler representation than a Beta distribution function, and Pohl and Chapman (1987).

Gamma Distribution [Cheng 1977] Used by Lootsma (1989) as it is more natural

and simpler than the Beta distribution function.

Beta Distribution [Rubinstein 1981, p81] Used in the popular PERT technique.

It should be noted that the networks have been assessed with a specific activity temporal function throughout as, more often than not, managers use a single function with which they are familiar. The affects of several different, combined activity temporal functions on a project duration is an area for future research (and represents an enormous sample set).

5.6 RESULTS FOR CONTINUOUS ACTIVITY TEMPORAL FUNCTIONS

5.6.1 Overview

In this section the results are presented for all possible configurations of activity networks with the particular activity temporal functions. The graphs shown represent histograms of the duration of the network concerned. In some cases a line graph is superimposed on the histogram to show the fit of a particular distribution function (identified by BestFit™). Normal probability plots of the network duration are also shown in some cases. In these results it is interesting to determine which distribution function provides the most accurate representation of the temporal function of the networks concerned.

5.6.2 Normal Distribution

In all cases the C-S and K-S tests for Normality did not fail for all network configurations. The C-S T-S values ranged from 0.00674 to 0.078, and the K-S T-S values from 0.020174 to 0.023013. There were two anomalies with the A-D test which provided T-S figures of 0.707188 (high precedence, high parallelism, ten activities), and 0.650595 (high precedence, low parallelism, ten activities). These figures were particularly insignificant when compared with other A-D T-S values obtained. Figure 5.8 shows a typical example of the results obtained. The line plot in this case shows a Normal distribution with mean 991, standard deviation 14.13.

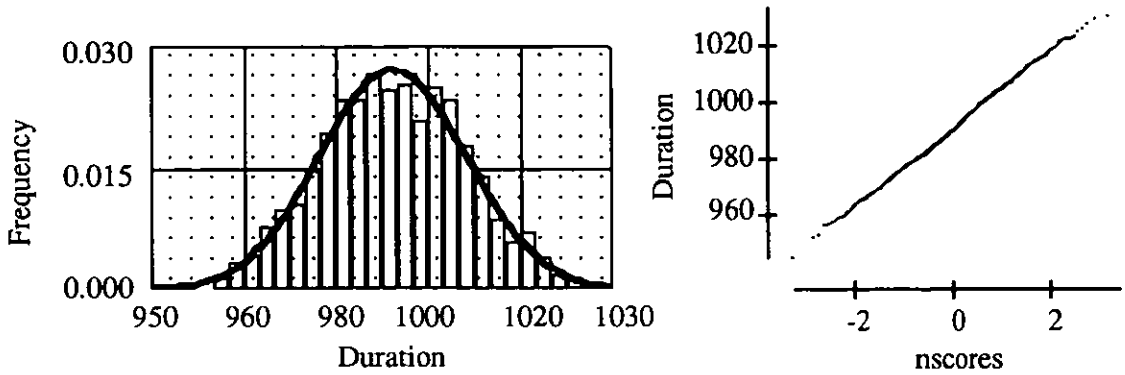


Figure 5.8 Normal, Low Precedence, High Parallelism, One Hundred Activities

5.6.3 Lognormal Distribution

All network durations prove to be distributed Normally as far as the C-S test was concerned (T-S values ranging from 0.00559 to 0.18194). For a network with high precedence and high parallelism both the K-S test and A-D test failed (having T-S values of 0.0794 and 8.9892 respectively). In this case the network duration showed signs of positive skew (skewed to the right). Although the C-S test for Normality did not fail (the T-S value was 0.18194) a more indicative fit was the Lognormal with mean 119 and standard deviation 3.39. Figure 5.9 shows this function plotted over the network duration histogram with these characteristics.

All other network configurations passed both the K-S and A-D tests for Normality. The K-S T-S values ranged from 0.0152 to 0.0234 and the A-D T-S values from 0.24319 to 0.62798.

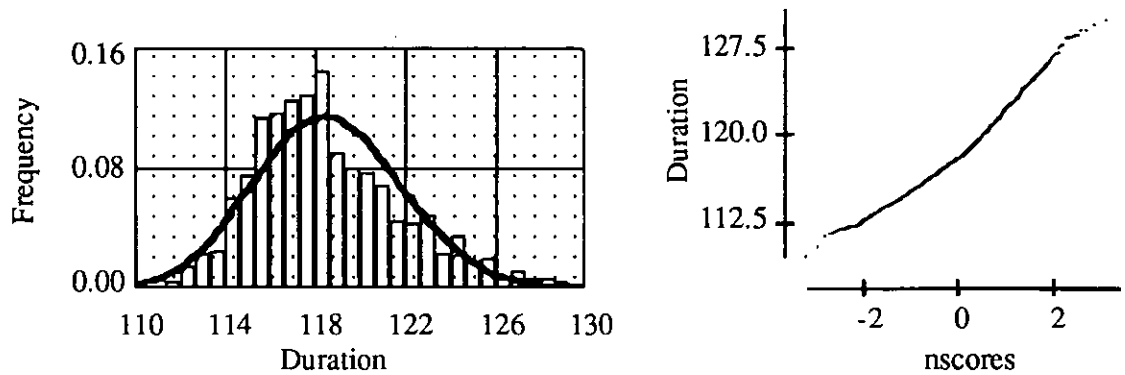


Figure 5.9 Lognormal, High Precedence, High Parallelism, One Hundred Activities

5.6.4 Rectangular Distribution

The results obtained for a network consisting of activities with rectangular temporal functions highlight the weakness of the C-S and K-S goodness-of-fit tests applied. Figure 5.10 shows a distinctive negative skew (skewness measured as -1.34) for the duration of a network with high precedence, high parallelism and ten activities. Even with such a distinctive shape the test for Normality did not fail (the C-S T-S value was calculated as 0.8635, the K-S T-S value as 0.1246). The Normal probability plot shown alongside is clearly curved. BestFit™ showed that a Beta distribution (α_1=2, α_2=0.53) was a more likely representation of this data having a C-S T-S value of 0.093092. Figure 5.11 shows a difference graph showing the absolute error between the network duration and this fitted Beta function. The A-D test did not accept the Normal Distribution as a good fit in this case (T-S value 28.94) preferring the fitted Beta distribution function (T-S value 2.439).

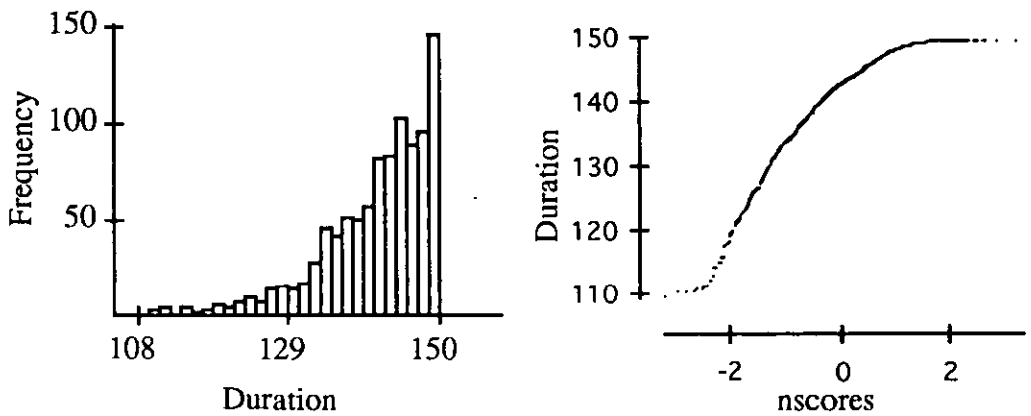


Figure 5.10 Rectangular, High Precedence, High Parallelism, Ten Activities

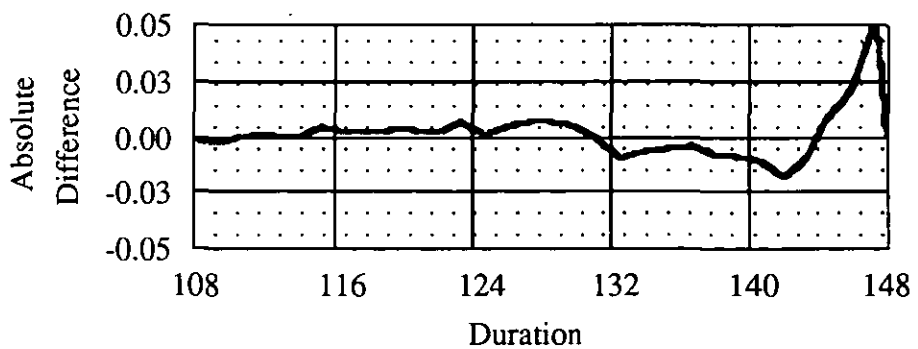


Figure 5.11 Difference Between Duration and Fitted Beta Distribution Function

Equally non-Normal network temporal functions were obtained for networks with high precedence and low parallelism. Figure 5.12 is particularly representative of these networks showing the temporal histogram of a network with ten activities. The C-S T-S value for the Normal distribution proves surprisingly low at 0.066616 in this case. BestFit™ indicated that a more likely representation would be a Beta distribution function (α_1=0.86, α_2=0.95) which had a C-S T-S value of 0.0192. Figure 5.13 represents a difference graph showing the absolute difference between the network duration and this fitted Beta function. This network failed both the K-S and A-D tests for Normality having T-S values of 0.1444 and 28.2 respectively. Both these tests favoured the fitted Beta function.

All other network configurations (those with low precedence) proved to be distributed Normally. C-S T-S values for these network durations ranged from 0.000789 to 0.000887, K-S T-S values ranged from 0.0190 to 0.0264 (placing the Normal distribution first), and A-D T-S values ranged from 0.3025 to 0.5180.

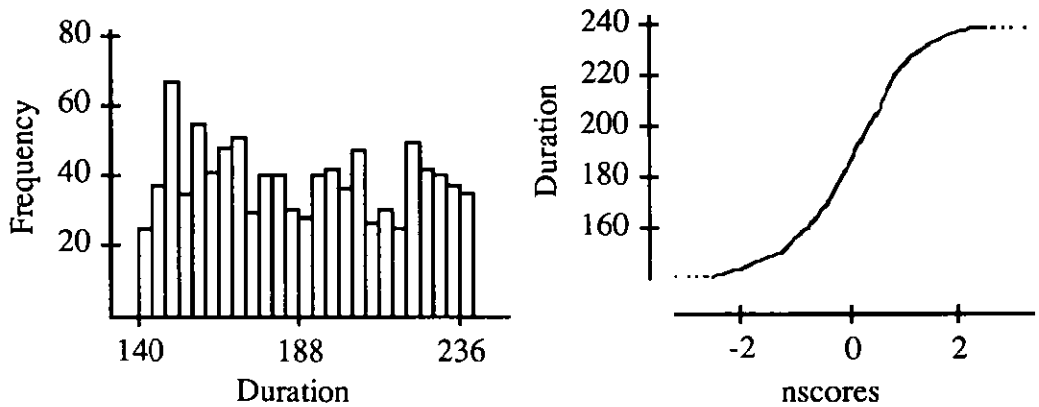


Figure 5.12 Rectangular, High Precedence, Low Parallelism, Ten Activities

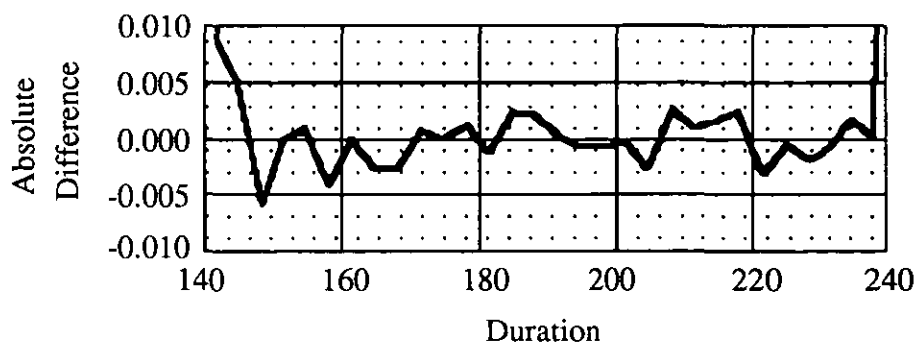


Figure 5.13 Difference Between Duration and Fitted Beta Distribution Function

5.6.5 Triangular Distribution

Activity networks with high precedence and high parallelism all showed signs of negative skew (for example, one with one hundred activities had a skewness value of -0.6363). The C-S goodness-of-fit test for Normality was still acceptable having a T-S value of 0.3458 in this case. BestFit™ identified that a more likely fit was the Weibull (α=106, β=147) with a C-S T-S value of 0.13346. This is shown in comparison with the network temporal histogram in figure 5.14. The K-S and A-D test were more sensitive in this case and, although identifying the Normal distribution as the best fit in both cases, failed to accept it as representative of the data having test statistic values of 0.05624 and 6.398 respectively.

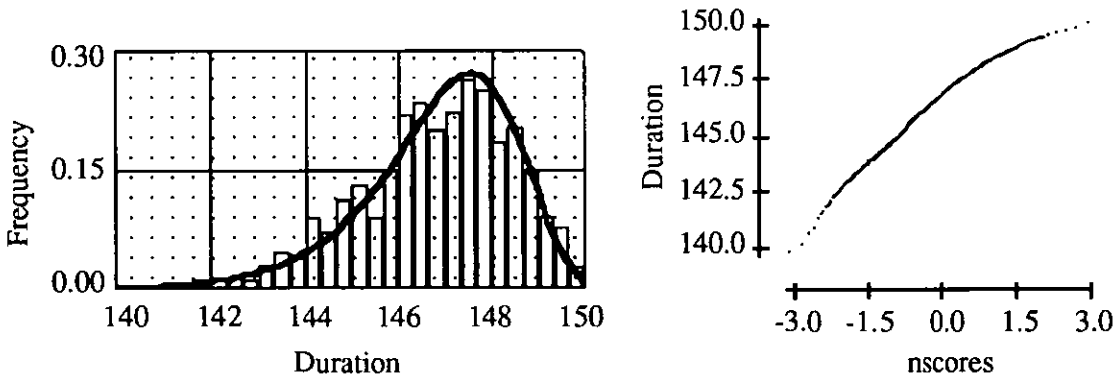


Figure 5.14 Triangular, High Precedence, High Parallelism, One Hundred Activities

For activity networks with a high precedence and low parallelism some more obscure temporal function shapes were obtained. In most cases the network duration is positively skewed (for example with ten activities the skewness value is 0.4752 (see figure 5.15)). In this case the most likely fit is a Triangular function (a=136, b=152, c=205) with a C-S T-S value of 0.010315. The Normal distribution comes out as the eighth most popular fit with a C-S T-S value of 0.051231. The K-S and A-D tests also prefer the fitted Triangular distribution, ranking the Normal distribution as the fifth most likely fit in both cases. A rather formless duration was produced with one hundred activities (figure 5.16). This network duration looks rectangular with some skew. In this case BestFit™ identified the most likely fit as the Triangular function (C-S T-S value of 0.057, a=116, b=181, c=200) with the Normal distribution coming in fourth place with a T-S value of 0.100529 (μ=172, σ=16.77). Figure 5.16 compares this network duration with both the fitted Triangular and Normal distributions. Both the K-S test and the A-D found the Normal distribution unacceptable having T-S values of 0.0880 and 13.26 respectively.

All other network configurations (those with low precedence) proved to be Normally distributed for all tests. C-S T-S values for these network durations ranged from 0.001615 to 0.001748, K-S T-S values from 0.186 to 0.2321, and A-D T-S values from 0.2955 to 0.58706.

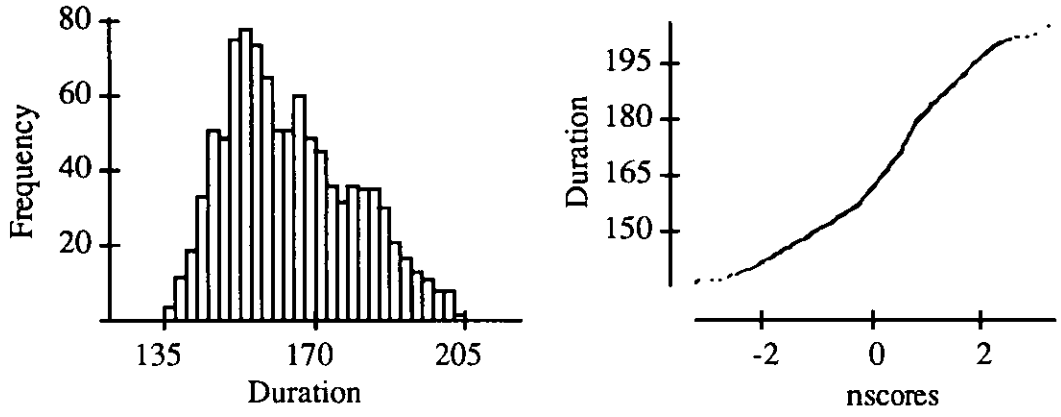


Figure 5.15 Triangular, High Precedence, Low Parallelism, Ten Activities

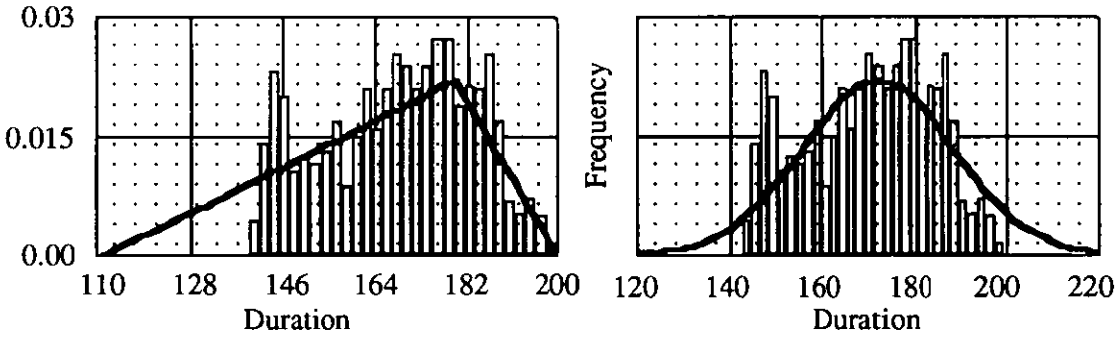


Figure 5.16 Comparison with a Triangular and Normal Distribution Function

5.6.6 Gamma Distribution

Although activity networks with a high precedence and high parallelism appeared to be skewed to the right (positively skewed) they did not fail the C-S test for Normality (C-S T-S value being 0.463126 for a network with one hundred activities). BestFit™ showed a more indicative representation would be the Logistic function (α=16.25, β=0.79) in this case having a C-S T-S value of 0.159439 or a Lognormal function (μ=16.53, σ=1.28) with a C-S T-S value of 0.179452. Figure 5.17 shows a comparison between

this activity duration and the fitted Logistic function. The K-S and A-D tests found the Normal Distribution unacceptable both preferring a fitted Lognormal distribution with μ=16.53, σ=1.28.

All other configurations of activity networks with Gamma activity temporal functions prove to be Normally distributed according to the C-S test which provided T-S values ranging from 0.006765 to 0.240463. The Normal Distribution was also ranked in first place by the K-S and A-D tests even though they would sometimes fail to accept any function at all. There were some anomalies with both the A-D and K-S tests for networks with high parallelism. In these cases the Normal distribution was ranked in fourth place although the differences between the T-S values and the first placed Lognormal distribution were quite negligible in these cases.

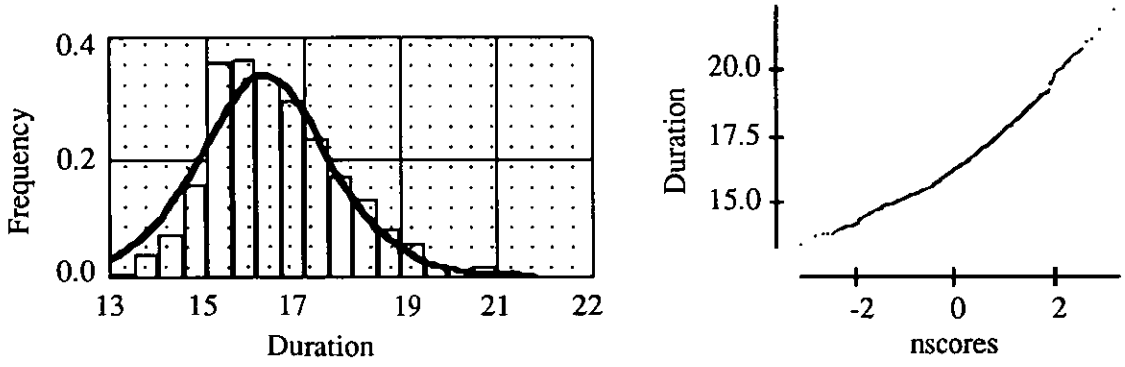


Figure 5.17 Gamma, High Precedence, High Parallelism, One Hundred Activities

5.6.7 Beta Distribution

All networks with a high precedence and high parallelism are skewed to the left to some degree. For example, the network with ten activities is shown in figure 5.18 (skewness value -0.827). In this case the C-S T-S for Normality was particularly high (16.44) yet within acceptable limits. BestFit™ indicated a more likely representation would be the Weibull (α=19, β=0.92) which was preferred by all three goodness-of-fit tests. A medium sized network (one with fifty activities) proved to be less skewed (skewness value -0.081071). In this case the most likely distribution fit was the Normal (μ=0.78, σ=0.0725) with a C-S T-S value of 1.38889. The K-S and A-D tests also preferred this fitted Normal distribution with T-S values of 0.0195 and 0.8003 respectively.

All other configurations of activity networks with Beta activity temporal functions proved to be Normally distributed according to the C-S test - having C-S T-S values

ranging from 0.111985 to 5.914. However, network durations with high precedence and low parallelism appeared triangular in shape with negative skew. For example, figure 5.19 shows the duration of a network with fifty activities (having a skewness measure of -0.64596) compared with the most likely distribution fit - a Triangular function (a=0.43, b=0.98, c=1.07) with C-S T-S value of 2.978881. The Normal distribution was the third most obvious fit in this case with a C-S T-S value of 6.156842.

For networks with high parallelism both the K-S and A-D tests accepted the Normal distribution as the most likely fit (for example, with low precedence the K-S T-S value was 0.0286, A-D T-S value was 0.8773) even though these figures were unacceptable at the α=0.1 level. For networks with low parallelism, the K-S and A-D tests were in conflict over which function provided the best fit, discounting all functions as unsuitable in most cases.

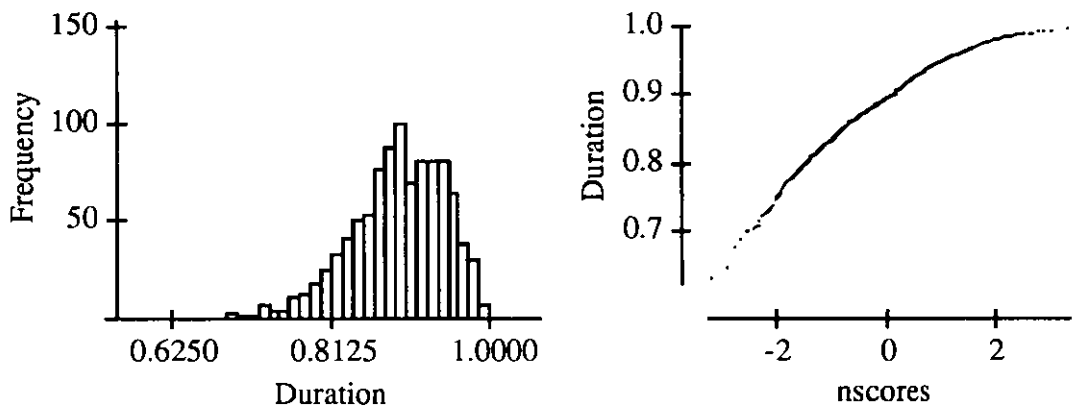


Figure 5.18 Beta, High Precedence, High Parallelism, Ten Activities

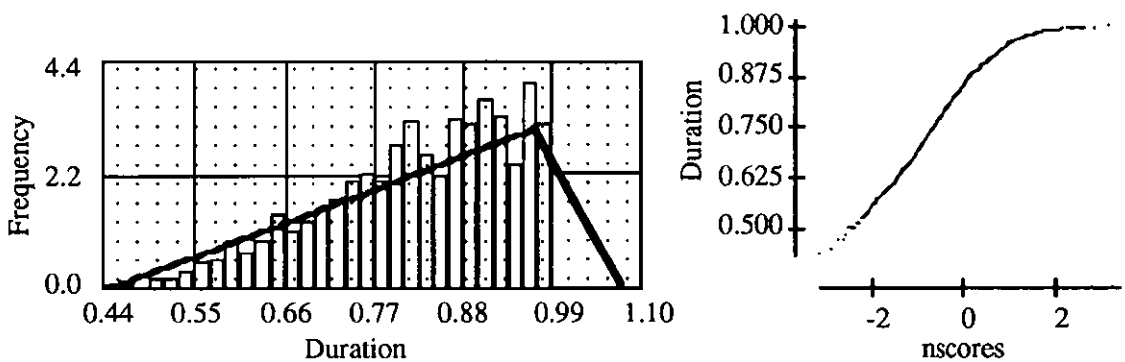


Figure 5.19 Triangular, High Precedence, Low Parallelism, Fifty Activities

5.6.8 Continuous Results Summary

The C-S test accepted all network durations as approximately Normal even in cases when a visual check would discount this claim. This emphasises the weakness of the C-S goodness-of-fit test in certain situations and shows that it should not be used in isolation. The K-S test only accepted Normality in 46% of cases whilst the A-D test was even more sensitive accepting Normality 32% of the time. If one looks more closely at particular network configurations, networks with low precedence are, on the whole, more likely to be Normally distributed, being so 62% of the time with both the K-S and A-D test. This is due to the additive nature of the temporal functions within the networks and stems from the Central Limit Theorem. In these cases the simulation algorithm was, in effect, taking N independent samples from a given distribution function. This is the basis of the Central Limit Theorem (for example, Schagen (1986)) which implies that the sampling distribution of the calculated network duration would be approximately Normal, especially if the sample size was increased towards infinity.

In cases of high precedence the results were clearly dependent on the temporal function of the activities themselves. The Central Limit Theorem in this case was overshadowed by the maximum combination of the activity temporal functions used. The maximum of several distribution functions does not approximate Normality as the results clearly demonstrate.

Networks consisting of activities with Normal activity temporal functions are seen to have an approximate Normal temporal function in all cases. This is not the case for activity networks with Beta activity temporal functions (such as PERT networks) where several different distribution functions appear to be more appropriate representations of project durations in some cases.

With low parallelism and high precedence the network temporal function generally approximated the activity temporal function used. With high precedence and high parallelism it was difficult to predict the outcome. In some cases the network temporal function was positively skewed (when the activity temporal functions were Gamma or Lognormal distributed). In other cases the network temporal function was negatively skewed (when the activity temporal functions were Triangular or Beta distributed).

In conclusion one can only rely on Normality for Low Precedence networks. In other cases one can expect the unexpected. The only guarantee in these cases is when the activity temporal functions are Normal throughout. In this case the network temporal function does tend to be Normal.

5.7 RESULTS FOR DISCRETE ACTIVITY TEMPORAL FUNCTIONS

5.7.1 Bernoulli Distribution

For all networks with high precedence and Bernoulli activity temporal functions, a single network duration figure was obtained. This figure represented the absolute completion time of these networks. It was generated because of the minimal variation produced between activity temporal outcomes. No test was applied to these results as their distribution characteristics were obvious.

For networks with low precedence and high parallelism several different network duration histogram shapes were produced. Figure 5.20 shows a network of ten activities with these characteristics. According to the C-S test, the best fit for this network duration was a Logistic function (α=9.81, β=0.64) having a C-S T-S value of 0.01931 (superimposed in figure 5.20). Both the K-S and A-D tests favoured a m-Erlang function (m=107, β=0.0939) having T-S values of 0.197566 and 54.73 (although this is quite clearly unacceptable) respectively. The larger network (with one hundred activities) was more accurately represented by a Binomial function (n=197, p=0.96) with a C-S T-S value of 0.015706 (figure 5.21). In this case the K-S test preferred a Logistic function (α=188, β=1.8) having a T-S value of 0.0769, and the A-D test preferred a Normal function (μ=188, σ=2.92) with a T-S value of 6.006. The C-S test accepted a Normal distribution as a possible fit in all these cases with C-S T-S values ranging from 0.00685 to 0.121.

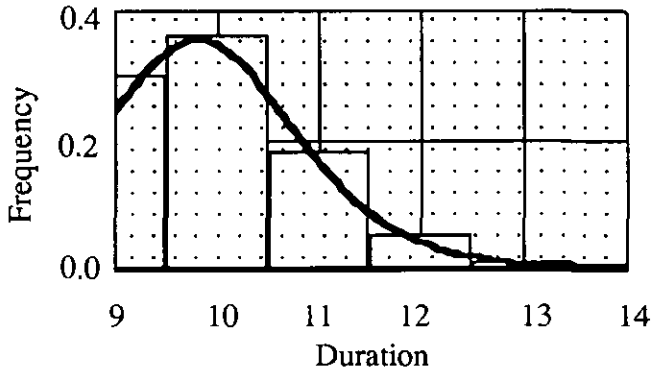


Figure 5.20 Bernoulli, Low precedence, High parallelism, Ten activities

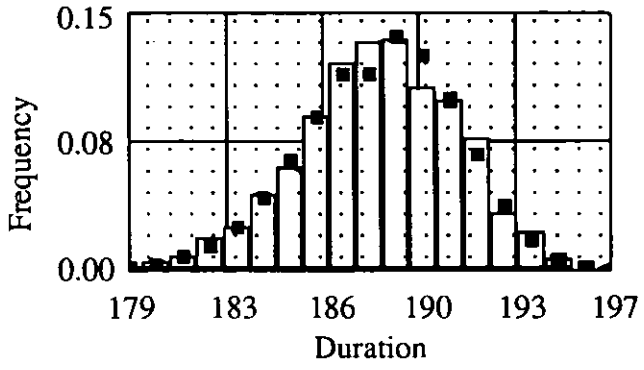


Figure 5.21 Bernoulli, Low precedence, High parallelism, One Hundred activities

For networks with low precedence and low parallelism there was again no definitive shape. A small network with ten activities (see figure 5.22) was best represented by a Lognormal (μ=10.77, σ=0.91) distribution function according to the C-S test with a T-S value of 0.049. Both the K-S and A-D tests preferred a m-Erlang (m=135, β=0.0798) function with T-S values of 0.2173 and 58.38 respectively. The C-S test also accepted the Normal distribution as a possible fit in all these cases having C-S T-S values ranging from 0.03291 to 0.112706. A network with these characteristics and one hundred activities is shown in figure 5.23. In this case a fitted Lognormal function provided the best fit according to the C-S test with a T-S value of 0.024447. Figure 5.23 shows this duration histogram with a fitted Normal function (the fourth most acceptable choice with a C-S T-S value of 0.03291).

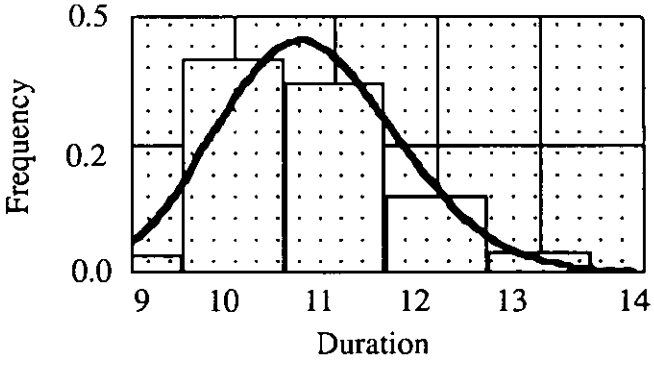


Figure 5.22 Bernoulli, Low precedence, Low parallelism, Ten activities

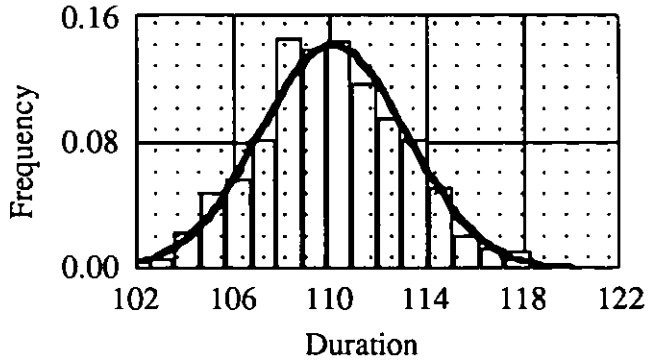


Figure 5.23 Bernoulli, Low precedence, Low parallelism, One Hundred activities

5.7.2 Binomial Distribution

Networks with high precedence and high parallelism produced no distinct temporal function varying from single durations to those shown in figures 5.24 and 5.25. According to the C-S test a network with only ten activities (figure 5.24) was best represented by a Gamma function (α=9.31, β=0.29) having a T-S value of 0.01399. A Normal distribution function was placed in ninth place by this test with a T-S value of 0.577. The K-S and A-D tests also failed to agree on a most likely function, favouring the m-Erlang (T-S value 0.2225) and the Weibull (T-S value 60.91) respectively. For larger networks (one hundred activities) with these characteristics (for example, figure 5.25) the Lognormal (μ=18.18, σ=1.53) was preferred by all tests. T-S values were 0.041, 0.1498, and 20.57 for the C-S, K-S and A-D tests respectively. The Normal function was still accepted by the C-S test in this case with a T-S value of 0.126508.

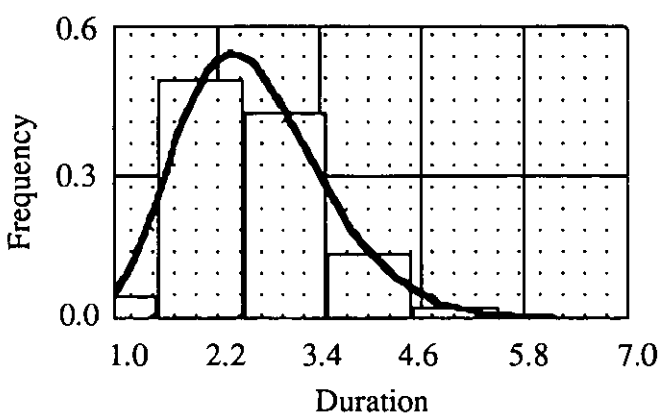


Figure 5.24 Binomial, High precedence, High parallelism, Ten activities

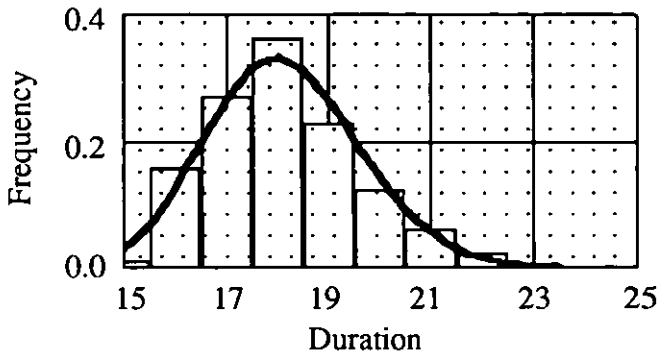
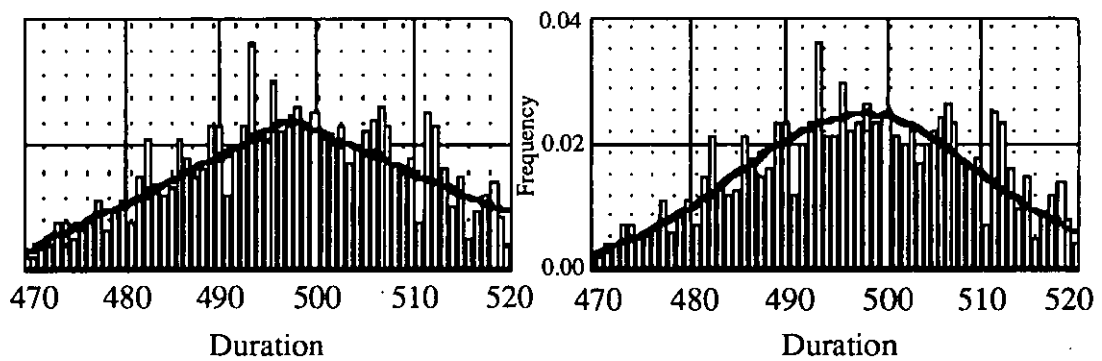


Figure 5.25 Binomial, High precedence, High parallelism, One Hundred activities

With low precedence and high parallelism more regular, almost rectangular shapes were formed. For example, figures 5.26 and 5.27 show the duration of a network with one hundred activities and these characteristics. The C-S test preferred a triangular distribution (T-S value 0.0844) as the best fit in this case (shown in figure 5.26) whereas both the K-S and A-D tests preferred a Normal function (figure 5.27) with T-S values of 0.03 and 0.087 respectively.



Figures 5.26, 5.27 Binomial, Low precedence, High parallelism, One Hundred activities

The networks with low precedence and low parallelism provided similar results. The Normal distribution function was preferred by both the C-S and A-D test (T-S values 0.028854 and 2.68 respectively). The K-S test ranked the Normal distribution as the second most obvious fit, preferring a fitted Logistic function.

With high precedence and low parallelism rather simplistic, minimal outputs were produced. Figure 5.28 is particularly representative of these results showing the duration of a network with ten activities and a fitted Normal function. The Binomial function (n=10, p=0.9) was the best fit according to C-S test (T-S value 0.0039). The

K-S and A-D tests both preferred a fitted Normal distribution function with T-S values 0.219 and 61.58 respectively.

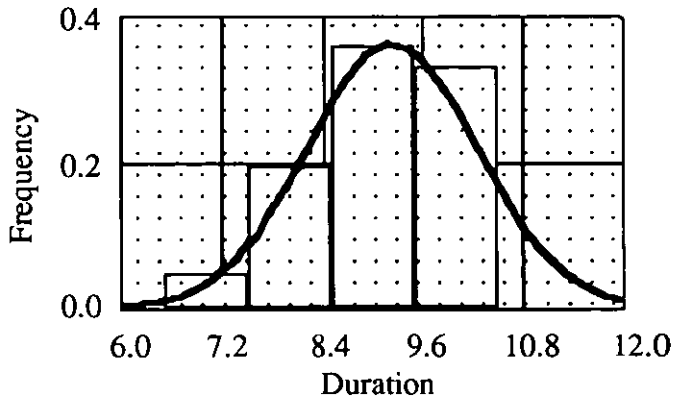


Figure 5.28 Binomial, High precedence, Low parallelism, Ten activities

5.7.3 Discrete Uniform Distribution

For networks with high precedence and high parallelism their durations were generally negatively skewed. For a network with ten activities (figure 5.29) both the C-S and K-S tests preferred a Weibull (α=15.16, β=10.47) as the most likely fit. The A-D test preferred a Normal function as the most acceptable fit although this was outside the acceptable bounds (T-S value of 76.173). The C-S value for the Normal in this case was still acceptable with a value of 0.2043. For a network with fifty activities (figure 5.30) the C-S test preferred a Weibull function with a T-S value of 0.06147. The K-S and A-D tests both preferred a Normal distribution with T-S values of 0.2314 and 52.091 respectively (figure 5.30 shows this Normal function superimposed on the duration histogram).

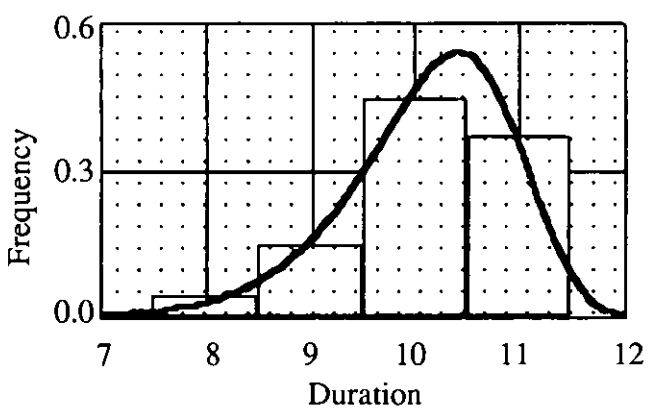


Figure 5.29 Discrete Uniform, High precedence, High parallelism, Ten activities

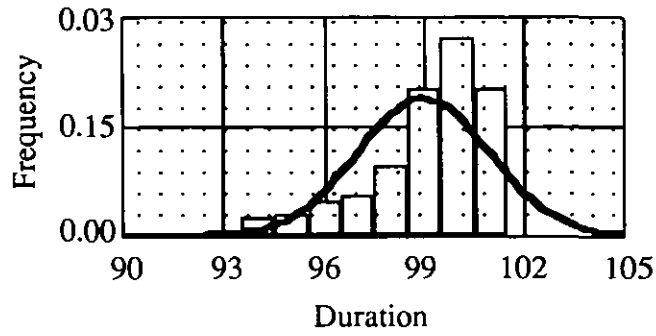


Figure 5.30 Discrete Uniform, High precedence, High parallelism, Fifty activities

All networks with low precedence showed signs of Normality - regular bell shaped curves. The C-S T-S values for a fitted Normal function ranged from 0.000249 to 0.12433, the K-S T-S values from 0.01304 to 0.01728, and the A-D T-S values from 0.21095 to 0.308934. Figure 5.31 shows a network with low precedence, low parallelism, one hundred activities, and a fitted Normal distribution function.

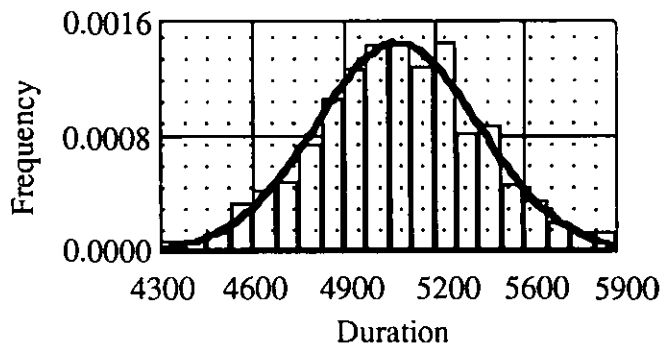


Figure 5.31 Discrete Uniform, Low precedence, Low parallelism, One hundred activities

On the whole the duration of networks with high precedence and low parallelism looked rectangular in shape. For example, figure 5.32 shows the temporal histogram of a network with fifty activities and these characteristics. In this case the best fit was a Beta function (α_1=0.65, α_2=0.85)*97+94 with as C-S T-S value 0.021103, or a Weibull(α=5.45, β=153) according to K-S and A-D tests (T-S values 0.07887 and 12.58 respectively). Figure 5.33 shows the absolute difference between the network duration, in this case, and the fitted Beta distribution function. The Normal function was still acceptable in these cases with C-S T-S values values ranging from 0.05363 to 0.2212.

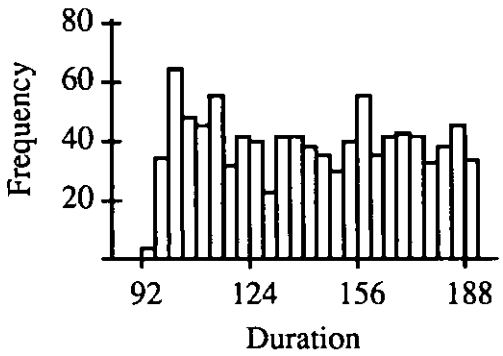


Figure 5.32 Discrete Uniform, High precedence, Low parallelism, Fifty activities

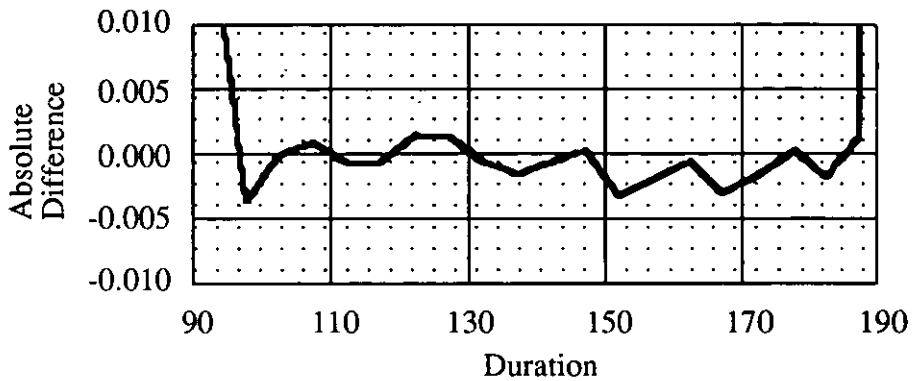


Figure 5.33 Difference Between Temporal Function and Fitted Beta Function

5.7.4 Negative Binomial Distribution

Networks with high precedence and high parallelism on the whole showed signs of positive skew. For example, a network with fifty activities is shown in figure 5.34 (skewness 1.021). In all three tests the best fit was identified as a Lognormal (μ=42.92, σ=11.75) (superimposed in figure 5.34) with a C-S T-S value of 0.0619, K-S T-S value of 0.0437, and A-D T-S value of 1.251. The Normal distribution function was acceptable according to the C-S test in these cases with T-S values ranging from 0.141327 to 11.32. It was not acceptable to the more sensitive A-D test, however, in one case having a T-S value as high as 27.1.

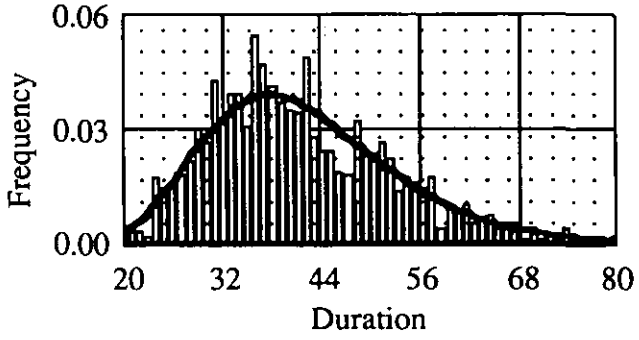


Figure 5.34 Negative Binomial, High precedence, High parallelism, Fifty activities

In all but one case of low precedence a reasonable bell shaped temporal curve was produced. For example, figure 5.35 shows a duration histogram of a network with one hundred activities and high parallelism. The C-S T-S value in this case was 0.000696. The K-S test accepted the Normal distribution as the best fit with a T-S value of 0.016255. The A-D placed the Normal in third place with a T-S value of 0.325044 (preferring a Gamma function in this case). In one case a duration with some positive skew was produced (figure 5.36 - low parallelism, one hundred activities, skewness 1.945846, with fitted Lognormal). A Lognormal (μ=13.9, σ=14.09) was identified as the best fit in this case by all tests with T-S values of 0.054018, 0.065109, 6.38671 for the C-S, K-S and A-D tests respectively. The Normal function was still acceptable to the C-S test with a T-S value of 0.3262.

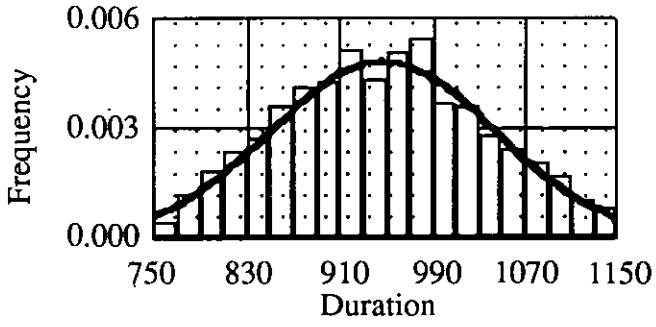


Figure 5.35 Negative Binomial, Low precedence, High parallelism, One Hundred activities

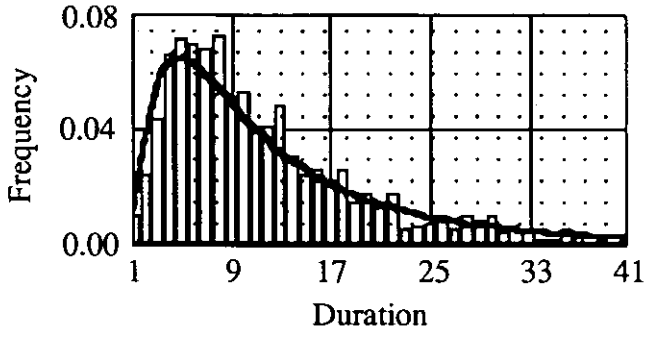


Figure 5.36 Negative Binomial, Low precedence, Low parallelism, One Hundred activities

With high precedence and low parallelism the networks showed distinctive positive skew. For example with ten activities (figure 5.37) no single fit was agreed by the tests. The C-S test preferred a Gamma (α=1.03, β=9.34) with a T-S value of 0.054992, the K-S test preferred a Lognormal(μ=10.88, σ=16.25) with a T-S value of 0.068284, and the A-D test preferred a Weibull(α=1.1, β=9.99) with a T-S value of 6.395508. This histogram again highlighted the weakness of the C-S test for Normality which, in this case, accepted Normality with a T-S value of 0.60648. The K-S was a little more sensitive with a T-S value of 0.340905 and the A-D provided a more understandable rejection with a T-S value of 138.332686.

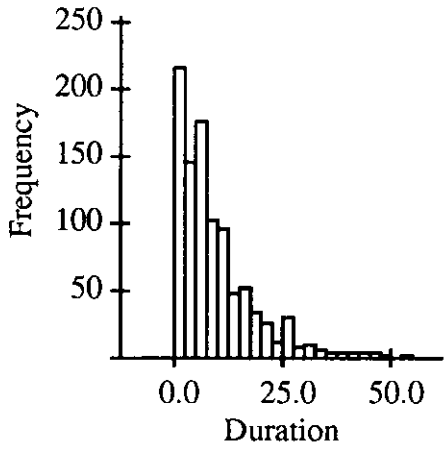


Figure 5.37 Negative Binomial, High precedence, Low parallelism, Ten activities

5.7.5 Poisson Distribution

All networks with high precedence and high parallelism showed signs of slight positive skew - for example figure 5.38 shows a network with fifty activities (skewness 0.573707). In this case a Lognormal function most favoured fit by all tests with T-S

values of 0.018812, 0.106772, 13.254991 for the C-S, K-S and A-D tests respectively. The C-S test for Normality provided acceptable T-S values ranging from 0.06754 to 0.229335, even though this function was rejected by the K-S and A-D tests in these cases (again highlighting the weakness of the C-S test in isolation).

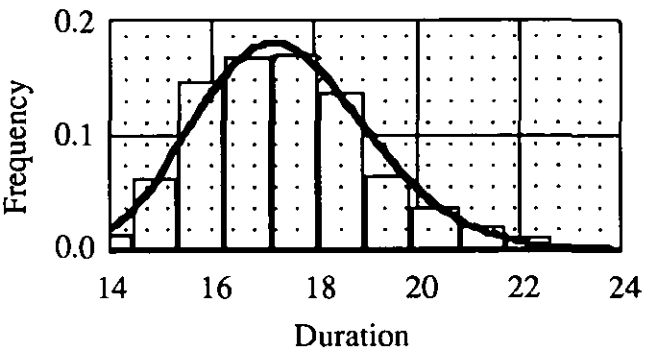


Figure 5.38 Poisson, High precedence, High parallelism, Fifty activities

All other network configurations, with Poisson activity temporal functions, appeared to be reasonably bell shaped. For example, one with low precedence, low parallelism and ten activities is shown in figure 5.39. In this case the Normal distribution function was accepted as the best fit by both the C-S and A-D test (T-S values of 0.036038 and 2.521598 respectively). The K-S preferred a fitted Logistic function but still placed the Normal distribution function as its second choice. For these network configurations the Normal C-S T-S values ranged from 0.034587 to 0.059232. The K-S and A-D tests were less convinced, however, accepting the Normal as the best fit in only two cases and a mixture of other distributions at other times.

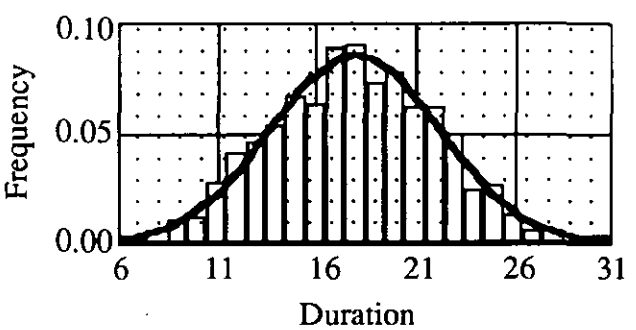


Figure 5.39 Poisson, Low precedence, Low parallelism, Ten activities

5.7.6 Discrete Results Summary

Although the results presented above appear rather different, they do provide some important conclusions. The first point to note is that one cannot assume that the duration of an activity network can be represented, in all cases, by a single distribution for any discrete activity temporal function. The C-S test would, however, discount this as it accepted all network durations as approximately Normal, even in cases when a visual check would discount this claim. This emphasises the weakness of the C-S goodness-of-fit test in certain situations and shows that it should not be used in isolation. One reason why the C-S test is particularly weak in this area is that it does not give great weight to the important tails of a distribution which can represent substantial project overruns.

In total, eleven different functions were selected by the three goodness-of-fit tests as representative of an activity network duration at some stage. From the fourteen functions that were assessed, only the Hypergeometric, the Geometric and the Chi-square distributions were not chosen as good fits at any stage.

Viewing the tests individually the C-S test preferred no outright function, selecting both the Normal and Lognormal functions 20% of the time. All other distribution functions were evenly spread throughout the remaining 60%. The K-S test also had no definite result, selecting the Normal as the best fit 28% of the time, the Lognormal and Logistic 18%, and the rest evenly spread. The A-D test was more decisive preferring the Normal distribution function 52% of the time, with all other functions evenly spread.

Overall, from all three tests, the Normal distribution function was the most popular fit (34% of the time), and the Lognormal distribution second (16%). All other functions were fairly evenly spread. These results were not unexpected when one looks at the shape of the temporal histograms produced. The duration of activity networks take various shapes - standard Normal bell shapes, positively and negatively skewed shapes, rectangular shapes, and single point values. In 80% of cases, although all durations were in discrete form, they were modelled more accurately by one of the continuous distribution functions (this, of course, could only be determined by the C-S test). This result must be treated with some caution as, after determining that activities and projects complete in whole time units, one would not use the real parameters provided by these functions. It is worth emphasising again that modal classes provide far more accurate estimates of project completion times and, for one-off projects (that most tend to be), represent the most likely duration.

5.8 CHAPTER SUMMARY

Many other distribution functions can be used to represent both the cost and duration of activities within a project. The results presented in this chapter show that managers should not use a single mean figure as representative of the overall completion time of a project but should view the resultant duration as a whole function. These results have shown that both Generalised and Probabilistic Activity Networks cannot be easily represented by a single distribution function and should be viewed on an individual basis. No distinct distribution function can be used to represent the duration of a project with any of the activity temporal functions analysed here. In conclusion, it makes far more sense for managers to use the modal class as an approximation of the expected project duration than a single mean value that could grossly underestimate or overestimate the duration of a project with a highly skewed, irregular or level temporal function. Cumulative distribution functions also provide more valuable results and this is emphasised by the availability of project management software packages with such facilities.

CHAPTER 6

An Artificial Intelligence Approach to Software Development Management

CHAPTER PREFACE

This chapter is presented in the form of a working chapter. It puts forward a proposal for an artificial intelligence based support tool for software development management, developed from the working paper of Dawson and Dawson (1993a). Blackboard Architectures are introduced that provide a framework on which a tool of this kind can be built. Reason and temporal maintenance systems (also known as belief revision systems) are also examined to provide a means of maintaining several project plans and providing several problem solutions simultaneously (if they exist).

CHAPTER KEYWORDS

Software Development, Project Management, Artificial Intelligence, Blackboard Architectures, Reason Maintenance, Temporal Maintenance

6.1 INTRODUCTION

6.1.1 Overview

Chapter two discussed in some detail the main components of the software development process and introduced a metamodel for controlling this process at the strategic level. An area in which there has been little research or development over recent years is the development of 'intelligent' software process management systems. In this chapter, Blackboard Architectures are introduced that can support the development of an intelligent software process management system. This system focuses on the problems associated with managing the development of software. As different project management problems are encountered at different stages of a development process, this chapter looks at artificial intelligence techniques that can be used in particular phases of this process. It is worth noting that while the responsibility for project decisions should and must lie with project managers, a knowledge based assistant could provide decision support for these and other activities [Ahmad et alia 1988]. The development of such a

system represents a significant input of research. This is clearly beyond the intended scope of this thesis and perhaps represents a doctoral work in its own right.

The software development process was identified in chapter two as a particularly complex task (complexity being identified by Brooks (1987)) as inherent within modern software systems). This complexity makes software systems notably difficult to build. Compounding software development problems are behavioural problems, such as those detailed by Kerzner (1989). Kerzner stated that the most common causes for overdue, and over cost projects, are behavioural.

In many respects the root cause of software development problems can be attributed to poor management of the software development process. There are two identifiable reasons for this. First, it is the responsibility of managers to ensure good team morale, human relations, and labour productivity. Without these qualities in a project, a software development team could soon develop the behavioural complications identified by Kerzner. Second, it is the responsibility of management to set realistic time, cost, and performance targets for a project. After all, when one identifies that a project is overdue and/or over cost, it may be that this is in respect to previous, possibly unrealistic goals.

It is the initial intention of this chapter to put forward a proposal to assist this second managerial responsibility - that of determining realistic and attainable goals within a software development process and identifying an optimum software development plan. Management inaccuracies in planning lead to projects that slip further behind schedule leading to low team morale and, as a consequence, augment the first managerial problem. Managers should also be supported throughout all phases of a project and not just within initial planning stages. Because different problems are encountered at different stages of a project development, this chapter looks at how artificial intelligence can support all these stages.

6.1.2 Context

Chapter two has already covered in some detail the aspects and approaches to software development. With reference to figure 2.2, this chapter presents a support element that assists a particular activity (project management) through various phases of a software development life cycle. By identifying particular artificial intelligent techniques within different phases of the life cycle, these techniques can assist managers through the development of a product.

6.1.3 Artificial Intelligence

Although the fields of artificial intelligence and software development are extremely diverse, there are specific artificial intelligence techniques that are particularly useful in certain aspects of the software development process. Applying artificial intelligence techniques to general project management is a relatively new field of research. Examples of more recent work include Kunz et alia (1986), who proposed the use of multiple worlds as a method of husbanding various project plans simultaneously. Several plans could be stored as *worlds* in this system (based on Assumption Based Reason Maintenance) and could be viewed, by a manager, to compare different project plan solutions (a kind of *what if* analysis). In Kunz et alia's example project this system enabled choices for the location of a graving dock to be viewed concurrently, based on factors including geology, site location and labour productivity.

In 1987 Foster [Foster 1987] looked at potential applications for artificial intelligence in more general project management, and in 1986, Sathi et alia [Sathi et alia 1986] developed Callisto - an intelligent project management system. The Callisto project devised intelligent project management tools for documenting expertise and exploring phases in the development of computer system prototypes. Callisto could be used to manage different objectives within a project - for example, resources, product configuration, and activities. This work also introduced a smaller prototype system called Mini-Callisto.

In 1987 Guerrieri [Guerrieri 1987] explained the application of expert systems to project management in a paper that included truth and temporal maintenance approaches. Guerrieri's work included project scheduling and showed how an explanation facility could be incorporated within Prolog. This facility explains how project decisions were obtained. Guerrieri also introduced *time guards* that provided a means of temporal maintenance within project plans. Also in 1987 Levitt and Kunz [Levitt and Kunz 1987] analysed the phases and levels of project management, before proposing the application of specific artificial intelligent techniques to these phases and levels. They also looked at the use of blackboard architectures and detailed the PLATFORM system, developed from KEE™ and the multiple worlds approach of Kunz et alia (1986). Chapman and Manesero (1988) developed an intelligent management system for use in the construction industry in 1988 and Noronha and Sarma (1991) provided a detailed study of artificial intelligence approaches to scheduling problems in 1991. Their work touched on the use of PERT and CPM as a means of project scheduling. Expert systems have also been used to assist management decision making from a higher integrative level. For example, Spangler (1991) presented a paper on how artificial intelligence could be used to assist the strategic

decision making process.

6.1.4 Project Management Software

Apart from the few exceptions highlighted above, the majority of software tools available in the project management market today do not incorporate any real artificial intelligence concepts. Most tools merely automate the calculations (resource levelling, critical path analysis and so on) involved in established techniques such as activity networks and Gantt charts. Although these tools provide more user friendly interfaces and rapid results by automating these calculations, they still leave any deliberation and *what if* analysis to the project manager. These software tools have been developed for support and, as stated by Plasket (1986), 'There is not a piece of software that will 'manage' your project; only you can do that'.

Generally speaking, project management systems employed by many companies for the development of software produce a single, baseline plan at the initiation stage of a project (usually at a tactical level). Any variations to this plan - such as adjustments to milestone dates, development techniques employed and so on - are applied to this baseline as anomalies. The metamodel defined in chapter two overcomes many of the problems associated with losing touch of an initial baseline by providing a flexible strategic level plan to start with. With a rigid baseline system a software development manager perhaps feels obliged to 'get it right first time'. Too many variations to an initial baseline can complicate a plan and lead to possible misinterpretations.

6.2 LIFE CYCLE PHASES

6.2.1 Overview

Chapter two identified a high level phase set that is applicable to all software development projects. By breaking the development process into these *master* phases (Analysis, Synthesis, Operation, and Retirement), specific managerial problems can be identified within each phase. For example, during the analysis stages difficult decisions need to made for the scope, size, resource requirements, and location of a project at various organisational levels. Within synthesis, more decisions need to be made as targets are missed and team moral falls. During operations, the costs and effects of various changes must be considered, and when a software system approaches retirement, decisions on the feasibility of replacing that system must be made.

Breaking these master phases down through an organisation's levels (strategic and

tactical) using a Work Breakdown Structure increases the level of detail and eventually identifies the tasks involved in the development of a software product. For the purposes of this chapter Analysis is broken down into Objective Setting and Planning. Control is identified as a task within the synthesis phase, and maintenance as part of operations. Figure 6.1 shows these *submaster* phases as distinct from the master phases. These submaster phases can be considered more simply as:

- Idea - Identify something that needs to be done
- Plan - Plan how to do it
- Do - Control the doing of it
- Improve - Improve and/or modify it

6.2.2 Objective Setting

Objective Setting represents the initial conceptualisation of a project where decisions are made as to the type, size, location etc of that project. This stage is sometimes referred to as Conceptual Design and is used to clarify the objectives of a project and to determine project priorities. Objective setting determines the aims and deliverables of a project.

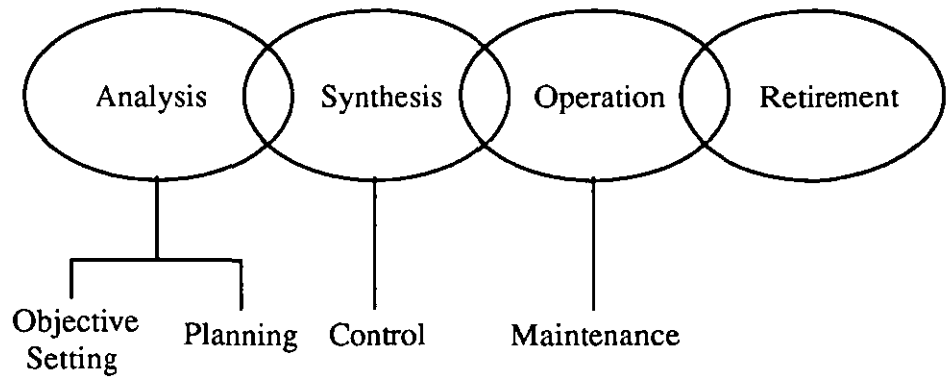


Figure 6.1 Phase Hierarchy

6.2.3 Planning

Planning defines the tasks necessary to complete a project with the assistance of a Work Breakdown Structure. At a tactical level, tasks are arranged in an ordered network using precedence analysis. During this phase management decides on a project's milestones that are included in a network plan. Incorporated into planning is the scheduling of durations, start and finish times, and resource requirements for each task. This was

covered in some detail in chapter four.

6.2.4 Control

Control can perhaps be viewed more as a managerial activity than as a submaster phase in its own right. It consists of tracking a project as it progresses and adjusting future expectations accordingly. Again, control was covered in more detail in chapter four.

6.2.5 Maintenance

Maintenance represents any post release work that may be needed after a system has gone live. This can include enhancing and upgrading software, programming software for use on other platforms, and producing more user friendly, faster enhancements. It can also include activities such as providing a help desk facility and removing any bugs that are not found until post-release.

Ideally, at this level, each phase should complete before the next one begins. In practice it is more likely that overlap occurs between phases and some form of feedback takes place (see chapter two). Scheduling pressures can also cause an overlapping of these steps (for example, Pulk (1990)). This overlap allows a developer to feed results back more easily to earlier phases and emphasises the need for a more flexible approach to the phased plan.

6.3 ARTIFICIAL INTELLIGENCE TECHNIQUES

6.3.1 Overview

Over recent years attempts have been made to incorporate some form of intelligence into various project management systems. Techniques have been applied specifically to different phases of a project life cycle - Objective setting, Planning, Scheduling and Control. One example is provided by Levitt and Kunz (1987) who proposed the following artificial intelligence techniques for each phase:

Objective Setting	Assumption Based Truth Maintenance System.
Planning	Means end AI planning and domain specific knowledge.
Scheduling	Knowledge based interactive graphics and knowledge processing.
Control	Knights and Villains and a knowledge processing system.

There are a broad range of management tasks operating at different organisational levels

and aimed at different project objectives. As these tasks operate through different phases of a development it is clear that a single artificial intelligence technique would need to be extremely flexible and powerful to be applicable to every software development process. Working with the blackboard architecture that is introduced below provides an artificial intelligent framework on which knowledge, appropriate to problems encountered at different stages and at different organisational levels, can be applied.

6.3.2 Blackboard Architectures

According to Adler (1992), a blackboard architecture is intended to address the following objectives:

(i) To incorporate diverse sorts of knowledge in a single problem-solving system.
(ii) To compensate for unreliability in the available knowledge.
(iii) To compensate for uncertainty in the available data.
(iv) To apply available knowledge intelligently in the absence of a known problem-solving algorithm.
(v) To support cooperative system development among multiple builders.
(vi) To support system modification and evolution.

Adler went on to explain how each of these factors can be achieved by blackboard architectures. By incorporating these capabilities, a blackboard architecture provides a powerful approach to problem solving in different problem domains.

'The Blackboard Model is a relatively complex problem-solving model prescribing the organisation of knowledge and data and the problem solving behaviour within the overall organisation' [Nii 1986a]. This reference to Nii provides one of the more detailed studies of blackboard architectures. Nii introduces the concept of this artificial intelligent structure and provides, in Nii (1986b), examples of systems that use this architecture. This text was based on the work of Hayes-Roth (1983, 1984, 1985a, and 1985b) who developed the concept of blackboard architectures and provided a comprehensive coverage of its applications.

The earliest reference to Blackboard Architectures can be traced back as far as 1962 when Newell wrote [Newell 1962]:

> 'Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it'.

This statement encapsulates the aspects of more up-to-date views of Blackboard Architectures. Blackboard architectures can be pictured in much the same way as ordinary classroom blackboards. A problem is written onto a *Blackboard Data Structure* (either by a user, or by an internal source within the architecture) and various experts (called *Knowledge Sources*) apply themselves separately to solve what they can of this problem. The order in which the knowledge sources approach the blackboard data structure is determined by a *Control Unit* that interprets what each knowledge source has to offer to a solution (the knowledge sources bid for a chance to work on the blackboard). The knowledge sources continue to solve what they can of the problem until either a solution is reached or more information is required for them to proceed. Figure 6.2, which illustrates this blackboard architecture, is adapted from Nii (1986a).

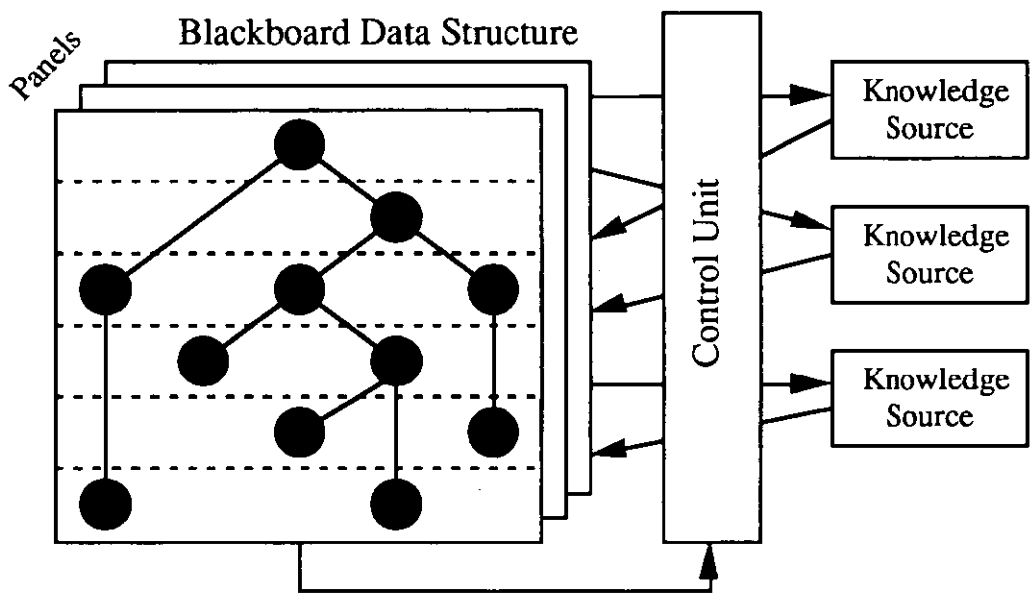


Figure 6.2 A Basic Blackboard Architecture [Nii 1986a]

A blackboard system therefore consists of three main components - a blackboard data structure, knowledge sources and a control unit.

(i) Blackboard Data Structure [Nii 1986a]

The blackboard data structure is a global database within the management system. It holds computational and solution state data needed by, and produced by, the knowledge sources. This data structure is partitioned into different levels of analysis that correspond to application-dependent hierarchies. The different levels of the hierarchy (represented by the dashed lines in figure 6.2) represent the different levels of object

properties that are connected by named links. Figure 6.2 also shows that a blackboard can be made up of several *panels* each of which can represent several different solutions to a problem. Contradictory information can be maintained concurrently by this architecture using reason maintenance systems that are discussed later.

(ii) Knowledge Sources

Knowledge needed to solve a problem is partitioned into knowledge sources that are kept separate and independent. In a project management system, for example, knowledge sources may be created that can perform critical path analysis, resource levelling and so on. The knowledge sources transform information on one level of the blackboard hierarchy into information on the same or other levels using algorithmic procedures or heuristic rules. These knowledge sources can work on several different panels of the data structure, providing several solutions to a problem. Knowledge sources can only interact with one another through the blackboard data structure.

(iii) Control Unit

The control unit decides which module of knowledge to apply next, based on the current solution state and what each knowledge source has to offer. This results in an incremental generation of partial solutions to a problem on a blackboard panel. Figure 6.2 suggests one way in which the control unit fits into the overall blackboard architecture. From this position it monitors the current solution state on the blackboard and manages the knowledge sources. It is possible that the problems encountered by the control unit (ie deciding what part of the problem to focus on and which knowledge source to apply next) can themselves be solved by a blackboard architecture. In this situation the control data can be incorporated into one of the blackboard structure panels where knowledge sources, specifically aimed at these problems, can work.

There are several variations to this basic principle including changes to the blackboard data structure itself, the knowledge sources or the control element [Nii 1986a and 1986b]. At some stage inconsistencies may appear on the Blackboard and some form of knowledge or factual maintenance is required. This maintenance can be performed by reason maintenance systems.

6.3.3 Reason Maintenance

Artificial intelligence systems need to maintain a model of their particular environment. The domain, that represents this environment, needs to be updated at various stages to reflect perceived changes in this environment. One reason for updating the model could be the discovery of contradictory information about the environment in the domain or the introduction of new information that contradicts facts already there. The

conventional approach for removing these contradictions is to change the most recent decision made. This is known as chronological backtracking and is the technique used in more popular artificial intelligent languages such as Prolog. An alternative solution is not to change the most recent assumption made, but to change the assumption that produced this unexpected condition. This alternative approach is called reason maintenance or belief revision. Broadly speaking reason maintenance exists in two different forms - Justification Based Reason Maintenance and Assumption Based Reason Maintenance. A good coverage of most of the literature in this field was presented in Martins (1990).

(i) Justification Based Reason Maintenance

Reason maintenance (and Reason Maintenance Systems) were originally introduced in a paper by Doyle in 1978 [Doyle 1978]. Doyle actually based his work on an earlier text of Stallman and Sussman (1977). In Doyle's initial work [Doyle 1978, 1979a, 1979b, McDermott and Doyle 1979 and 1980] reason maintenance was referred to as truth maintenance which was a rather confusing terminology since it was not truth that was been maintained but the reasoning behind assumptions. Strictly speaking his approach is a Justification Based Reason Maintenance System (a JRMS). It has several disadvantages when compared with the Assumption Based Reason Maintenance System (ARMS) that was developed by DeKleer in 1984 [DeKleer 1984] (see below).

A Justification Based Reason Maintenance System maintains one consistent database at a time and will facilitate switching out of that database if it becomes inconsistent. Rather than backtracking chronologically, it will employ *dependency directed backtracking* so the source of an inconsistency is rapidly isolated and removed. A JRMS allows non-monotonic justifications, unlike the initial ARMS, but it only allows beliefs to be changed if a contradiction is found within the knowledge base. Each statement or rule within a knowledge base is represented by a *node* [Doyle 1979b]. Nodes can either be IN (believed to be true) or OUT (not believed to be true). Attached to each node is a list of *justifications* (hence JRMS) validating that node. There are two kinds of justification - *support lists* and *conditional proofs*.

A support list provides a list of statements or rules that, because they are deemed to be true or false, justify a particular node. Conditional proofs, on the other hand, represent hypothetical arguments that represents an implication of some facts. The main drawback of the JRMS is its inability to maintain several possible solution states simultaneously. Another problem is that changing one set of beliefs into another only occurs when a contradiction is detected [Martins 1992]. In *what if* analysis this can result in delays as previously computed solutions need recalculating for comparative purposes.

(ii) Assumption Based Reason Maintenance

An Assumption Based Reason Maintenance System is based on manipulating assumption sets rather than justifications. The ARMS processes multiple contexts simultaneously and consequently has the advantage of making available all possible solutions or partial solutions to a user. However, according to Dressler (1988) an ARMS only allows monotonic justifications (ie once information is added it cannot be removed for that particular program run).

Exploiting an ARMS allows a problem solver to work efficiently on all solutions simultaneously and avoids the computational expense of backtracking [DeKleer 1984, 1986a, 1986b, Reiter and DeKleer 1987, Martins 1992]. When a contradiction in knowledge is detected, all assumptions underlying that assumption are directly identifiable. This removes the need for backtracking that could remove current, possibly acceptable, assumptions.

Work by DeKleer and Williams in 1987 [DeKleer and Williams 1987] identified some advantages of reintroducing backtracking into an ARMS. They stated that three problems associated within an ARMS are:

(i) The task may require only a fraction of a search space to be explored.
(ii) Even for problems where all solutions are required, an ARMS would often search more than was necessary.
(iii) They are inherently more difficult to debug.

As an ARMS only allows monotonic justifications, there are advantages to be made by combining both an ARMS and JRMS together. Examples of combining these techniques include Rodi (1989), Dressler (1988) and Urbanski (1988).

6.3.4 Temporal Maintenance

Another problem associated with maintaining information within a knowledge base is that an environment can change over a period of time. The reason maintenance systems introduced above cannot cope with this temporal variation and therefore another form of reason maintenance - temporal maintenance - is required. Within a project management system, time represents an important factor (for example, it is one of the project objectives identified by Turner (1993)). As project conditions are dynamic, and results are constantly changing with respect to time, some form of temporal maintenance is required.

Temporal maintenance systems keep track of the consistency of a knowledge base at a

given time or time interval. In reason maintenance systems updates are usually required as more information is contributed to the system or contradictions are discovered. In temporal maintenance systems changes can be seen to occur over a period of time alongside more usual information modifications. The system is aware that the knowledge base is affected by the passage of time and updates it accordingly [Guerrieri 1987, Dean and McDermott 1987, Allen and Hayes 1985, Tsang 1988, Shoham and McDermott 1991].

Within temporal maintenance systems current valid states are based on underlying valid assumptions (the same as reason maintenance) that can, in this case, include temporal parameters. Rather than absolute dates being used, a temporal maintenance system can use reference intervals [Allen and Hayes 1985]. This implies that much of the knowledge within these systems is organised relatively rather than absolutely. Examples of these relationships, from Allen (1983), include X *before* Y, X *equal* Y, X *meets* Y, X *overlaps* Y, X *during* Y, X *starts* Y, and X *finishes* Y. This form of temporal relationship appears to relate directly to the relationships represented within activity networks. Guerrieri (1987) employed temporal maintenance in his system that dealt with multiple projects concurrently. This worked by applying *time guards* to facts within the knowledge base to identify the interval in which these facts are held to be true.

As temporal systems change assumptions based on the passage of time, they must also be able to access previous information that has since become outdated (in case this information needs reworking for any reason). This is particularly important in project management where reference to an original, baseline plan may be required. In addition, the model must also be able to support persistence. In other words, if something has happened to produce a change in state from S1 to S2, it will remain in state S2 until there is further information or a progression of time to indicate otherwise.

6.3.5 Interaction

The question must be raised as to how the reason and temporal maintenance systems can interact with one another as data is manipulated on a blackboard. The obvious solution is that temporal maintenance is kept within each context of the ARMS. If this were not the case, temporal inconsistencies would be difficult to spot unless there was some form of temporal overlap between each context. A solution to this problem is to build temporal maintenance into the knowledge sources themselves. As the knowledge sources only ever see a single context at any given viewing, so would the temporal maintenance system. Interaction between the components of the temporal maintenance system within the knowledge sources is required in this case to ensure that temporal

consistency is maintained throughout the whole of the blackboard structure, and not just in a specific knowledge source region.

6.4 THE INTELLIGENT SOFTWARE DEVELOPMENT SYSTEM

6.4.1 Overview

Having outlined some approaches that are used in artificial intelligent systems, this section provides some initial thoughts on how these techniques can be applied to support the management of software developments. Section 6.2 showed how the development of software can be decomposed into four project submaster phases that present different managerial problems. Each of these submaster phases can be helped by different types of knowledge captured within an artificial intelligent system.

6.4.2 The Blackboard Architecture

Project managers possess a vast range of skills and knowledge that they apply to problems encountered in the development of projects. In order to pull together the diverse knowledge and skills for a software development process some form of structure is required. The blackboard architecture introduced earlier represents a suitable framework for this task. Not only does it allow separate knowledge sources (and consequently approaches) to work on particular problems, but it also structures the problem development in a hierarchal way that enables a software development plan to be split logically into constituent phases or levels. There are several aspects that require representation on a blackboard - the organisational levels, the different phases through which a project progresses, the different problem solutions, and perhaps even different organisational objectives if these are kept separate. How these aspects are represented within a blackboard data structure, that has only three possible dimensions (hierarchical, panels, and reason maintenance supporting multiple solutions in these dimensions) is a question that still needs addressing.

Initial thoughts on this representation are to use the different panels of a blackboard data structure to store separate solutions to a project management problem. Each panel of the blackboard, in this case, could represent a possible project scenario. For example, separate routes through a metamodel at the strategic level could be represented on different panels of the blackboard. The hierarchical levels of each panel could be used to represent different organisational levels of a project plan - for example, the top level could represent the integrative level, the next level down a strategic plan (metamodel level) and the next level a tactical plan. A knowledge source for this structure could

perform a work breakdown structure on the strategic level plan to produce lower hierarchal levels. Knowledge sources could also be used to provide different forms of *what if* analysis with the solutions kept on different panels of the blackboard. This representation does not account for the different phases through which a project progresses and the different objectives that may be involved.

An alternative view could be to use the hierarchy of a blackboard panel to represent plans for different phases of a software development process. Knowledge sources for different stages of the development process could then provide solutions to problems encountered at each stage. In this case each panel could represent the different levels of organisational interest (integrative, strategic, and tactical) or different problem solutions encountered within each phase. Obviously there may be several solutions for different problems and these need maintaining simultaneously by some form of reason maintenance in particular areas of the blackboard data structure.

6.4.3 The Knowledge Sources

Initial ideas in this area are to group the knowledge sources into phase specific sets so that each set produces alternative solutions to a problem within each development phase. The way in which these alternative solutions are maintained is discussed for each specific phase below. These knowledge sources represent knowledge that is captured from particular fields of project management. Initial ideas for knowledge sources include critical path analysis, resource levelling and ones that can implement some form of *what if* analysis.

Acquiring knowledge for knowledge based systems is by no means a simple task. Various texts have been written on the subject such as Hart (1989). There are also several approaches to representing captured knowledge within a knowledge source. This knowledge is usually represented as rules within the knowledge source that 'fire' when they can solve a particular problem presented to them on a blackboard panel. This usually occurs when facts on the blackboard match the antecedents of a rule within a knowledge source. Approaches to knowledge representation are studied in more detail in Ringland and Duce (1988) and Brachman and Levesque (1985) to name but two.

The following sections discuss the use of reason maintenance as a technique that assists the knowledge sources in providing solutions to problems encountered in different phases of a project. As discussed in section 6.4.2, these different phases could be represented on different hierarchical levels on a blackboard panel. Each panel in turn could represent either different problem solutions or different organisational levels of interest (in which case multiple solutions would exist at each hierarchical level).

6.4.4 Objective Setting

During this submaster phase the objectives of a project must be set in line with the organisation's objectives (see chapter four). To assist the management of this particular phase some form of decision support tool is required. At this stage of a development tradeoffs must be made between scope, time, and cost while there is still considerable uncertainty about other detailed development parameters [Levitt and Kunz 1987].

An Assumption Based Reason Maintenance System applied to this stage of the development would allow managers to analyse the scope of a development by comparing different outcomes concurrently. Underlying assumptions could be reviewed and revised using the ARMS to assess alternative approaches generated by different knowledge sources. The ARMS would be maintaining alternative project objectives that were devised by specific knowledge source sets. The ARMS is an ideal means of knowledge maintenance in this stage because the architecture is particularly well suited for tasks where a reasonable fraction of the potential solutions must be explored [DeKleer 1986a].

6.4.5 Planning

In some respects this stage is similar to that of objective setting. Alternative solutions are sought to determine the best balance of time, resources, and costs for a development.

Planning is more analytically based with solutions being derived through various rules and heuristics within the knowledge sources. There is not one heuristic that produces an optimum plan under all circumstances and consequently the knowledge sources capture alternative plan solutions that can be viewed by management.

The multiple worlds approach, developed by Kunz et alia (1986), is particularly well suited to this stage. Again this is based on an Assumption Based Reason Maintenance system and allows multiple solutions of a problem to be viewed and analysed concurrently. The way the worlds are created enables them to be maintained as complete units, so that reworking is not required each time a manager wishes to view an alternative plan. This allows a more real time, interactive management process, providing a manager with 'online' comparisons.

For the knowledge sources this also provides a practical technique. Different plans can be assessed by weighing up the tradeoffs between resource, cost, and time levels. Consequently the knowledge sources themselves can perform trial and error approaches (*what if* analysis) based on perturbations or changes that they introduce into plans on

the blackboard. Feedback from these changes provide knowledge sources with information they need to deduce the sensitivity of specific parameters.

6.4.6 Control

This is an ongoing activity throughout the working stages of a development. It is affected primarily by the advancement of time in which tasks begin and end. Consequently a temporal maintenance approach is proposed to control the changes that occur within this stage. Temporal maintenance updates the consistency of the knowledge base as time progresses. As tasks starting and completing significantly affect a project's status, and these events depend on time, the temporal maintenance approach is ideally suited to this particular stage.

The relative way in which the knowledge is represented within a temporal maintenance system allows the system to update dynamically a project task plan as it progresses. Anomalies which are input by a manager affect current and future progress. These changes cascade through the knowledge base via the relative connections imposed by the temporal maintenance system.

The knowledge maintained by a reason maintenance system within a blackboard can also be *time guarded* [Guerrieri 1987]. This means that each fact or rule is augmented with *time guards* that allow information to be believed or disbelieved at a particular time or time interval. These guards could be defined with relative, rather than absolute, temporal values because a software development plan is itself a relative structure. How *time guards* can be implemented within a relative data structure is an area for future research.

6.4.7 Maintenance

As highlighted in previous work this can be the most costly stage (with respect to both time and resources). It tends to be, however, the one in which the least planning and control is used. Initially, in the maintenance stages, strategic decisions must be made as to the depth and quality of future work that may be required for a specific software product. This could be assisted through a similar approach to that used during the objective setting stage.

It is important that managers extend the enthusiasm they have for project planning into this maintenance stage. They should weigh up alternative approaches to specific tasks in this stage with as much care as they used in objective setting.

An ARMS can be used to assess alternative cost/resource requirements and productivity from proposed work involved. It can also be used as a decision support system to test the feasibility of alternative updates that may seem beneficial to a development in the future. A decision support system based on an ARMS would assist management, for example, in assessing the feasibility of debugging a system, recalling software and introducing enhancements.

6.4.8 System Inputs and Outputs

An important question that must be raised is what are the intended inputs and outputs from this type of management system? At the initial objective setting and planning stages this does not pose a problem. It is envisaged that from an initial project outline (possibly at the strategic level) the system could apply work breakdown structure rules to decompose a project to lower tactical levels. It would also provide various project scenarios that could be compared, either by managers, or by knowledge sources. However, what are the inputs and outputs from knowledge sources within control and maintenance? One envisages an intelligent system that suggests changes in the project management approach as information is fed into the system based on a project's progress. These ideas are still in their infancy and require maturing before their actual implementation can be achieved.

6.5 CHAPTER SUMMARY

6.5.1 Conclusion

The theories and ideas introduced in this chapter are still very much in their infancy. They represent initial thoughts, and a possible solution, to the problem of developing an intelligent project management support tool that can be used in the development of software systems. There is obviously much work to be done before these thoughts and ideas can mature into a fully operational system at any level. The Assumption Based Reason Maintenance System has been identified as tool that allows managers to view several solutions to management problems concurrently and assess the alternatives. Temporal maintenance has also been identified as a technique that is directly applicable to dynamic changes encountered by projects.

6.5.2 Future Work

Although there is much work to be done in developing the ideas put forward in this chapter, there is already a blackboard framework in place at Loughborough University

on which the LUMP (Loughborough University Manufacturing Processing) system performs process planning [Herbert et alia 1990]. As this system is domain independent, the knowledge sources for the software development system can be applied directly to it. Work is still needed to capture the knowledge used for managing the software development process into the knowledge sources of this system. The initial steps would be to incorporate simple project planning knowledge sources into this system. Determining the final hierarchical/panel structure of the blackboard also requires addressing, as does the development of temporal maintenance within this system.

CHAPTER 7

Summary, Evaluation, Conclusions

CHAPTER PREFACE

This chapter brings together the work covered within this thesis. It summarises each chapter in turn, identifying advances made within those chapters, and highlights areas of potential future research. Current industrial approaches to software development are studied and real projects are examined where advances made in this thesis can be applied. Project management information systems, currently available within industry, are also surveyed.

KEYWORDS

Metamodels, Project Management, Generalised Activity Networks, Temporal Analysis, Artificial Intelligence

7.1 INTRODUCTION

The aim of this chapter is to collate the work of this thesis. With reference to figure 2.2, a support element has been developed within chapter three that supports the project management activity within a new software development model (chapter two). Chapter four put this work into context and improved the approach of activity network analysis, and chapter five used this development to analyse potential projects' temporal outcomes. Chapter six represented a working chapter, again providing a support element for project management of the software development process.

This thesis has covered several areas, all intertwined within the management of the software engineering process. This conclusion will start by summarising each chapter in turn before looking at the current state of the software development industry and examining projects where some of these ideas can be applied. Current industrial practices are also assessed to see where the concepts introduced here can be applied.

7.2 CHAPTER TWO

7.2.1 Summary

Chapter two (based primarily on the work presented in Dawson and Dawson (1994c)) began by identifying an early concept of the software development process, its aspects, and how these aspects interacted. This concept was developed and brought up-to-date by introducing identifiable models and methodologies. Having identified the essential aspects of the software development process, this chapter went on to develop a new concept called metamodels. Metamodels were defined as combined paradigms or hybrid models allowing more flexible, visual development of software systems - enabling software to grow within their structure. The benefits of metamodels were identified and a means of control was introduced (Generalised Activity Networks). The chapter concluded by identifying two areas of potential research:

(i) Development of software dependent metamodels.
(ii) Development of organisation specific metamodels.

In the following section another potential research area is identified - that of the interaction between established methodologies (for example, SSADM) and new, more dynamic approaches to the development of software systems.

7.2.2 Discussion

Many companies have recognised the need for more dynamic approaches to software systems development. A concept paper published by the Butler Group in Autumn, 1993 [Butler Group 1993] identified, once again, the need for more flexible approaches to the development of software (see also Agresti (1986a and 1986b)). Although keen to push forward their own concepts on software development management, they highlighted the problems encountered by companies using engineering approaches to this different management field. Their methodological approach is aimed at managers involved in strategic level planning of information systems developments. Figure 7.1, taken from the Butler Group report, emphasises the problems encountered by software development managers who use old engineering practices.

The Butler Group emphasised the need for a closer interaction between developers and client. Too little feedback between client and developer can lead to a system being incorrectly specified and produced. Too much feedback can lead to an overly complex system produced months (or even years) late. Figure 7.1 shows how, at the initial stages of a development, the user does not really know what they want. If they are then

excluded from feedback in subsequent stages they will not get what they really want in the final delivery. The cost of errors discovered in each phase is also shown to increase dramatically over the life cycle of a project. Usually one sees a ten fold increase in error costs for each subsequent life cycle stage as each stage inherits problems from earlier mistakes. By limiting software developments to out-dated, life cycle models, based on engineering practices, these are the results that ensue.

The main point from this report is that development of software is a dynamic process, constantly changing to reach either varying customer goals or a more comprehensible problem domain. The metamodel, introduced in chapter two, provides a hybrid model that can cope with more dynamic systems development. It provides a means of controlling software development projects and identifies risk and decision points within a project life cycle.

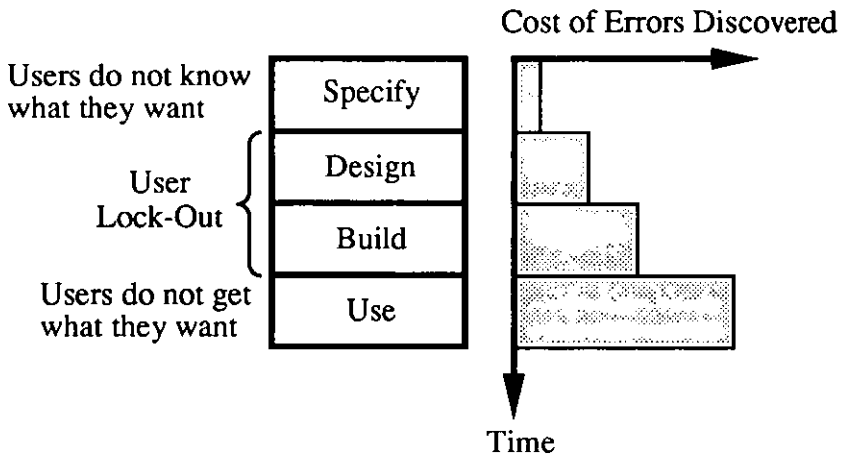


Figure 7.1 Flaws in the Engineering Model [©Butler Group 1993]

'Far from being the helpful tools they promised to be, software methodologies could turn out to be too complicated for their own good' [Davidson 1990].

Another point worth noting within current industrial practices is that, very often, projects are constricted, not just by models employed, but by development methodologies used. In a recent case at the University of Derby, a one person-year project was undertaken using SSADM. SSADM is, quite clearly, a methodology aimed at medium to large scale information systems developments. To apply SSADM in depth, as it was in this case, to a small project was unsatisfactory. The project was constricted by the limitations imposed on it by this methodology and was not completed successfully. It is important to realise that developments and models should not be dictated to by older development methodologies which are, after all, intended to support and improve the

development process.

One question that this raises is how new, flexible models, such as the metamodel developed in this thesis, are to interact with current, established methodologies. Although Generalised Activity Networks (developed in chapter three) provide a means of supporting such a model they do not answer the question of how methodologies could be adapted to cope with such flexible systems. Clearly, for smaller methodologies, covering only single stages of the life cycle (for example, data flow analysis, JSD and so on), this does not pose a problem. It is the larger, project-wide methodologies that require some adaptation. SSADM, for example, can prove quite constrictive in software development projects and the question posed should not be how metamodels interact with established methodologies but how these old methodologies can be adapted for newer, more dynamic systems development. The development of new, flexible methodologies, and the interaction between dynamic systems development and old methodologies, is another area for future research.

7.2.3 Evaluation

How a metamodel relates to a real software development project is discussed in some detail in section 7.7.4.

7.2.4 Conclusion

Chapter two introduces the metamodel which provides a new concept for model development within the field of software engineering. This flexible, dynamic structure provides a greater visibility of project direction and progress and provides a model that does not constrict the software development process.

7.3 CHAPTER THREE

7.3.1 Summary

Chapter three was based initially on work presented in Dawson and Dawson (1994a). It identified limitations in an area of project management support - Generalised Activity Networks - and began by identifying previous work in this field. Two project management software tools, that have limited abilities in this area, were also discussed. The early parts of the chapter concentrated on the work of Dawson and Dawson (1994a) - that of the development of a Generalised Activity-on-the-Arrow representation. The chapter moved on, in light of further research into current practices

in the project management industry, and developed a Generalised Activity-on-the-Node representation from this work.

An example of how Generalised Activity Networks could be used (more especially in software development management) was shown with a project developed for illustration. This example illustrated a tactical level plan for the implementation of a software system. Network properties were then discussed in view of current practices and approaches in this field. The chapter concluded by identifying four potential areas of future work:

(i) Implementation of the definitions in a software tool.
(ii) Gantt chart representation of Generalised Activity Networks.
(iii) Cost/Time optimisation and resource levelling in Generalised Activity Networks.
(iv) Work Breakdown Structures for Generalised Activity Networks.

7.3.2 Evaluation

Generalised Activity Networks provide more realistic representation of project scenarios. However, although some minor developments have been made in this field over the years, Generalised Activity Networks never became an established project management technique. A possible reason for this, which also makes evaluation of Generalised Activity Networks difficult, is that projects can always be seen to complete in a deterministic fashion. In other words, a project will complete at a given time along with the activities that constitute it. Probabilistic branches will have either been pursued or not, and loops performed a specific number of times. To say that a probabilistic activity that was not performed could have been omitted from a project plan would be wrong as, at the inception of a project, the final project environment can only be guessed at. At a project's outset a deterministic conclusion cannot be predetermined and only a probabilistic structure is able to represent the possibilities.

Generalised Activity Networks have already been successfully used in large development projects (for example, Moeller (1972), Kidd (1990)). The need for more flexible, network-based planning techniques can be implicitly identified by looking at how a software development can be successfully modelled by a metamodel (in section 7.7.4).

Many project managers interviewed felt that more powerful planning techniques, that include an ability to plan for alternatives, were needed. Generalised Activity Networks satisfy these requirements and support the management of more dynamic systems development.

7.3.3 Conclusion

The Generalised Activity Network representation provided in chapter three provides a conclusive definition in an area sadly overlooked in the past. The Generalised Activity-on-the-Node representation provides a support tool with which most project managers should be comfortable. The more powerful planning techniques (loops, probabilistic branching and so on), identified as a requirement by many project managers interviewed (see section 7.8.1), are provided as and when required. By encompassing all possible variations in one conclusive definition, a baseline has been set from which future developments can be made.

7.4 CHAPTER FOUR

7.4.1 Summary

Having defined a more powerful, new approach to project planning in chapter three, the aim of chapter four was to identify and improve analysis approaches for this and related techniques (ie PERT type analysis). Chapter four began by putting into context the uses of activity networks within the field of project management. It then moved on to a comprehensive coverage of past and present approaches to activity network analysis before concentrating on Monte Carlo simulation. A new approach was then introduced (a Dynamic Sampling Technique) based on work presented in Dawson and Dawson (1993b and 1994b). Antithetic variables (a less refined approach than stratification or Latin Hypercube) were also discussed as these can improve the efficiency of network simulations in certain cases (ie when activity temporal functions are symmetrical).

Limitations were noted with the Dynamic Sampling Technique and a prototype Modal Class Dynamic Sampling Technique was introduced. This prototype is in its infancy and requires development to overcome the problems identified with it in chapter four. The chapter concluded by noting four potential areas of research:

(i) The development of Modal Class Simulation of Activity Networks.
(ii) The affects of early delays on future activities.
(iii) Representing activity durations by more applicable distribution functions.
(iv) Parallel algorithms for the reduction of simulation costs still further.

7.4.2 Evaluation

Of all the risk analysis packages on the market today, none offer the ability to limit, by

accuracy, the number of Monte Carlo simulations performed. All the companies involved with these packages (for example, Predict!™, @Risk™ and Monte Carlo™) recommend rather subjective estimates for the number of samples needed to provide 'accurate' estimates of project completions and risks. None of these packages seem to identify a need to highlight the most likely project outcome (the mode) preferring to calculate the mean (which only represents an average if a project is performed many times). Nurse (a User Champion with Marconi) uses Opera® which is a Monte Carlo risk analysis package that accepts plans from Open Plan®. He claims that approximately one thousand simulations, based on a subjective estimate, should performed with this package to provide the required accuracy for network plan results. According to Nurse, one way of reducing the time for these simulations is to remove some of the lower risk activities beforehand (a kind of path deletion approach - see section 4.4.7). Some other project managers interviewed (see section 7.8.1) welcomed the modal approach and said they would find it of far more benefit than the approach offered by current risk analysis packages.

7.4.3 Conclusion

The techniques developed in chapter four provide more accurate and realistic representations of project outcomes in shorter times. These techniques now need integrating into Monte Carlo based systems. Managers can then reap the benefits offered by this, more efficient, approach.

7.5 CHAPTER FIVE

7.5.1 Overview

Chapter five used some of the techniques developed in chapter four to provide a detailed temporal analysis of activity networks. This chapter presents results from four papers [Dawson 1994a and 1994b, Dawson and Dawson 1993b and 1994b]. The results are presented as the comparative simulation requirements of Probabilistic and Generalised Activity Networks, and the affects that various discrete and continuous activity temporal functions have on the duration of projects represented by activity networks. It is unlikely that a single distribution function (for example, the Beta) can represent all activity temporal functions in all projects. The affect that various other distribution functions have on the duration of a project was, therefore, important to determine. The project outcomes determined in chapter five highlight the possibilities that can occur.

7.5.2 Conclusion

The one important conclusion that comes out of this chapter is that projects, when analysed by simulation techniques, cannot be represented by one particular temporal function. It is important that managers realise, having identified and incorporated risk into their project plans (both temporal and probabilistic) that project outcomes are not deterministic. In other words, project results must be viewed on an overall basis and statistical figures (for example, the mean and standard deviation), must be viewed with some caution. This work emphasises the need for modal estimates and cumulative frequency distributions that provide far more useful information to managers.

7.6 CHAPTER SIX

7.6.1 Summary

Chapter six represents a working chapter based on work presented in Dawson and Dawson (1993a). Overall it looked at how artificial techniques could be used to improve support for software project managers. It viewed management of the software development process from the angle of four phases and identified problems encountered by managers in these phases. A blackboard architecture was discussed that represents an ideal framework on which to build a software development management system. How knowledge from the field can be incorporated into the system as knowledge sources was also discussed. Reason maintenance systems were also introduced and a way of allowing interaction between reason and temporal maintenance systems within the blackboard architecture was proposed. The chapter concluded by looking in more detail at the artificial intelligence approach and its structure for a software development management system. It finished by identifying this work as an area for future development.

7.6.2 Conclusion

Artificial intelligence techniques for software development management are particularly limited at the moment and represent a target for the future. Artificial intelligence approaches to the more general field of project management are, however, more widespread. Examples include Levitt and Kartam (1990), Brown (1988) and Foster (1987). The approach put forward in chapter six provides a suitable framework on which intelligent systems can be developed. These systems can relieve some of the burden on the project manager by providing decision support where it is needed. At the moment, however, intelligent software development management systems are a future

goal and even then they may not be expected to manage projects on their own - only managers can really do that.

7.7 PROJECT MANAGEMENT SOFTWARE TOOLS

7.7.1 Overview

One cannot conclude a thesis that has developed new concepts in the field of project management without looking at some of the tools managers are currently using within both academia and industry. The best forum to assess these different tools was at a recent exhibition in London [Project Management South 1994] where all the companies active in this field were demonstrating their wares and introducing their own approaches to project-based management.

Turner (1993) identified five types of Project Management Information Systems (PMIS), three of which are available and two of which still require development. They are, respectively:

(i) Networking systems.
(ii) Cost and resource management systems.
(iii) Application generators.
(iv) Capacity planning systems.
(v) Totally integrated, modular packages.

Of these five systems, networking systems and cost and resource management systems are the two most closely related to the work of this thesis (mainly chapter three). Networking systems represent the more well known project management packages available today that include the ability to model projects by activity networks and/or Gantt charts (also called bar charts). The concepts within these networking packages were developed within chapter three. Cost and resource management systems represent a more methodical approach to project management and incorporate cost, work, and organisational structures within project plans. Generally speaking, cost and resource management systems represent more organisation-wide approaches to project management.

The following list presents the tools/methodologies/consultancies and companies active in the project management information system's market today. Most of these companies were presenting their tools at Project Management South (1994). Addresses for some these companies can be found in Appendix G.

Company	*Software Tools/Interest*
Aran Ltd	PMSX-Kernel™ (Ingres based)
ASTA Development Corporation Ltd	PowerProject™ Version 2
Baesema	RISKNET™
Complete Project Management Ltd	TrackStar™
Computer Associates® PLC	CA-SuperProject®
Computerline Ltd	PLANTRAC-APROPOS®
Deepak Sareen Associates	Time Line® 6, On Target™, InstaPlan™, ProjectGuide™
Hoskyns Group Plc	PMW, Project RISK
IBM (UK) Ltd	MITP
JMCA (John Cockerham and Associates)	Risnet™
LBMS	Methods On-Line, SSADM Engineer
Lucas Management Systems	Artemis Schedule Publisher™, and Prestige™
Leach Management Systems	CS Project™ for Windows™
MANTIX Systems Limited	Cascade®
Micro Planning International Ltd	MICRO PLANNER® Version 6
Microsoft®	Microsoft® Project™ 4.0
OPL	RISKMAN
Palisade Corporation	@Risk™
Panorama Software Corporation Ltd	Panorama™ PLANNER and COST
PARISS Ltd	PARISS Enterprise™
People in Technology Ltd	Pertmaster Advance 2.4G
Primavera Systems Inc	Primavera Project Planner®, Monte Carlo™ 2.0, Parade®
PSDI (UK) Ltd	Project/2 Series X®
Risk Decisions Ltd	Predict!™
Scitor Corporation	Project Scheduler 6™
TBV Consult	Consultants (see appendix A)
Welcom Software Technology Int.	Texim Project™, Open Plan®, Opera®, and Cobra®

7.7.2 Networking Systems

The majority of tools on offer were PC-based project management tools, now supporting the Windows™ environment. Generally speaking, they provide a means of planning projects using Gantt charts and activity networks (mostly Activity-on-the-Node). One tool, ProjectGuide™ (1993), provides a means of generating project plans

based more on an organisation's own approach or methodology. These plans can then be imported directly into CA-SuperProject®, InstaPlan™, Project Manager Workbench, Microsoft® Project™ or Time Line®. Most tools have an ability to level resources to some degree and some offer methods for project control (for example, earned value). Costs can also be applied and, in most cases, work breakdown structures can be created. Some of the tools allow data to be exported into databases or spreadsheets where other tools (for example @Risk™ (1990) or RISKNET™ (1992)) can be used to perform risk analysis on project plans. Those tools that perform risk analysis are discussed below.

The following list identifies the tools that fall into this category - the marketing details of which are beyond the intended scope of this thesis.

Panorama™ PLANNER and COST, Texim Project™, Artemis Schedule Publisher™, Primavera Project Planner®, Pertmaster Advance 2.4G, Project Scheduler 6™, Microsoft® Project™ 4.0, Time Line® 6, On Target™, CS Project™ for Windows™, PowerProject® Version 2, CA-SuperProject®, InstaPlan™, MICRO PLANNER® Version 6.

Turner (1993) identified that a requirement missing from current Project Management Information Systems is a completely integrated, modular system. He went on to state that perhaps the closest any company comes to this requirement is the suite of programs sold by Welcom Software Technology (Open Plan®, Cobra®, Opera® and Texim Project™). Turner also identified Cascade® (1993) by MANTIX Systems Limited as a system fulfilling this requirement, although in this case it is a single package with these facilities.

7.7.3 Cost and Resource Management

Large cost and resource management systems tend to be organisation-wide products that introduce a methodological approach to project management within an organisation. Two of the more well known packages that fall into this category are Cobra® (C/SCSC Cost Management) and Cascade®. Other systems, that are not as large but incorporate recognised cost control methodologies, are MICRO PLANNER® X-Pert (based on control using earned values) and InstaPlan™ (based on C/SCSC).

7.7.4 Methodology Based

Several companies at Project Management South offered methodology based tools (see also Powell (1990)). These packages either represent methodologies in their own right

(identified in figure 2.2) or support elements for already established methodologies. The better known of these tools are Cobra®, InstaPlan™ (both based on C/SCSC cost management), and Cascade®. Others include PMSX-Kernel™ and PLANTRAC-APROPOS® (both based on the Prince methodology), MITP (an IBM methodology), Methods On-Line and SSADM Engineer (methodology support by LBMS), MICRO PLANNER® X-Pert (based on control using earned values), and RISKMAN (a methodology based on a European Project Risk management initiative).

7.7.5 Risk Systems

'How can one manage and plan for the unknown when the "known" is always changing?' cited in Peltu (1994).

Perhaps of more interest to this thesis were the tools offering some form of risk analysis. Articles detailing risk analysis software include Computing (17 June, 1993) and Milton (1994). Of the thirty five tools listed earlier only six offered limited project risk facilities.

Opera®, by Welcom Software Technology, is an extension to Open Plan®. It provides four activity temporal functions - Normal, Uniform, Triangular and Beta and performs a Monte Carlo simulation to determine the mean and standard deviation of start dates, finish dates, and float. It also determines cost curves, and produces cumulative frequency distribution functions for times and costs along with histograms of these parameters. As it imports plans from Open Plan® it does not provide any probabilistic branching. The number of simulations performed is based on a subjective estimate recommended by the suppliers. This appears to be the case for all other Monte Carlo packages on offer today.

Predict!™ includes health and safety risks, financial appraisal, and marketing analysis. It provides an Activity-on-the-Node project representation but no probabilistic branching. It offers twenty two activity temporal functions and the ability for the users to customise their own. This system takes no bold steps in the field of project-based management but enhances concepts already in existence.

Another tool making no great improvements but advertising its developments nevertheless is Project Scheduler 6™. Project Scheduler 6™ allows three time estimates for each activity (as in PERT) and identifies the best, most likely and worst project completion time based on these figures.

Project RISK by Hoskyns provides a new concept in risk analysis. It represents a

question and answer based risk analysis system that provides no project planning. This system asks a user a set of questions to determine the risks involved with a project. It then reports back on various aspects of the project including the chances of completing the project, its tardiness, usability, and supportability.

Monte Carlo™ 2.0 and Risnet™ were two tools introduced in chapter three. Monte Carlo™ 2.0 is a plug in to Primavera Project Planner®. It allows two probabilistic branches, and two activity temporal functions. Risnet™ is an Activity-on-the-Arrow tool with limited node logic and twelve activity temporal functions.

@Risk™, supplied by the Palisade Corporation, is a plug in to either Lotus 123™ or Microsoft® Excel™. As projects planned using Microsoft® Project™ can be imported into Excel™ spreadsheets these plans can, in turn, be analysed by the @Risk™ package. @Risk™ provides either Monte Carlo or Latin Hypercube simulation methods (Latin Hypercube is a more efficient sampling technique for some distributions - see chapter four) that provide a form of project temporal risk analysis. As the results are based on plans drawn up within Microsoft® Project™, they are based on Probabilistic Activity-on-the-Node networks. Uyeno (1992) looks at the uses of @Risk™ in decision support for project managers.

RISKNET™ is another package that integrates itself within a spreadsheet. In this case it is the Smart spreadsheet and database manager. RISKNET™ was developed by the Anglo-French avionics group Baesema as part of their own risk analysis for Ministry of Defence contracts. The tool is limited to a Triangular activity temporal function based on three time estimates (most likely, pessimistic and optimistic). Although it then simulates a project to provide more detailed results, it can perform a deterministic analysis to resolve the project on a one-off basis. This form of analysis represents a concept identified in chapter four of this thesis - that of determining a most likely project outcome. RISKNET™, in this case, only performs one iteration of a project plan and accepts this result as the most likely outcome. In fact, what the system should do is to repeat the process several times and then conclude that the most most likely outcome is the one that occurs most often (the mode or modal class). The first result generated by RISKNET™ is not necessarily representative of this case.

In a recent survey, published in Personal Computer (May, 1994), @Risk™, Predict!™, RISKNET™ and Project RISK were assessed. Unfortunately, none of these packages incorporate probabilistic risk into project plans (unlike Generalised Activity Networks) and merely identify risk as a temporal or costing problem. The only two packages available on the market that incorporate some form of explicit risk within project plans are Monte Carlo™ and Risnet™. Risnet™ is based on the less familiar (certainly in the

1990s) Activity-on-the-Arrow representation and this, and Monte Carlo™, provide only limited probabilistic branching. The approaches of Generalised Activity Networks, such as those used in VERT and GERT, that have proved themselves in large, organisation-wide projects (for example Kidd (1990)) need reintroducing to industry today. The shortfall in potential tools with these capabilities is perhaps due in part to a case of managers not knowing what is possible and what they are missing.

7.7.6 Company Wide

Some of the project management systems introduced above represent larger, company-wide approaches to project management. Two of the more popular approaches are TrackStar™ (a matrix organisation tool providing consolidation and information to any management reporting level), and Cascade® that was covered in more detail in chapter four. Other organisation-wide approaches included Project/2 Series X® (PSDI (UK) Ltd), PARISS Enterprise™, Artemis Prestige™ (Lucas Management Systems), PMSX-Kernel™ and Open Plan®.

7.7.7 Summary

'Tools are not flexible enough to do what I want' cited in Peltu (1994).

The overall feel from industry at the moment is that project management tools are not flexible enough to cope with problems encountered in real projects. Although the companies offering tools in the current market are striving to do more than their competitors, very few provide a means of identifying and controlling risk. To make more impact than a rival, companies emphasise their system's user-friendliness and detailed reporting capabilities.

In some cases encountered, project managers are clearly hampered by the project management tools they are currently using. For example, figure 7.2 is an example of a particularly unclear Activity-on-the-Node plan produced by Microsoft® Project™. This is a plan of the development of a software system that monitors babies' conditions in neonatal care. It is a six person-year project developed by a small software house in Loughborough and Edinburgh (five staff spread between these two sites). The manager, in charge of this particular project, has very little interest in the plans produced due to their lack of clarity and detail. The project is pursued on a more informal basis as these plans provide no workable information from which to operate. The version of Microsoft® Project™ being used is an old DOS-based package, the more modern Windows™ versions providing more workable plans (for example, see the CA-SuperProject® example in Appendix A). Other project managers interviewed are also

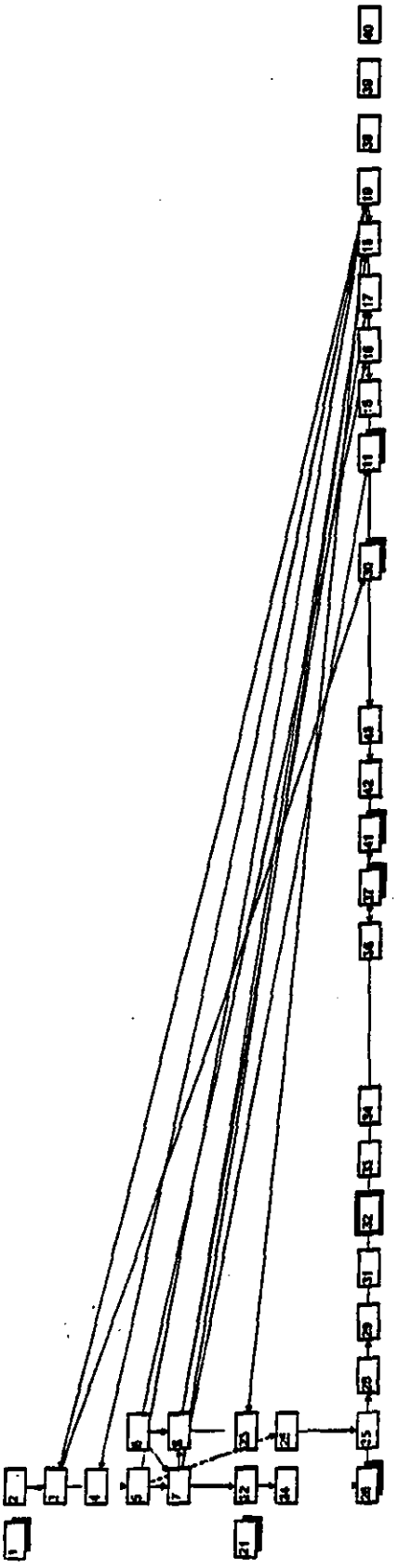


Figure 7.2 An Obscure Project Plan of a Real Software Development

looking for more powerful tools with clearer reporting capabilities. A representative of the RAF at Project Management South (1994) was looking at improving his project planning capabilities and was assessing Primavera Project Planner®. His current system was proving unworkable and a more powerful system, with a clear management reporting facility, was required.

In conclusion, the current state of project management information systems on offer are based on old, 1960s project management concepts (for example, PERT). Very few companies have taken bold steps to implement new ideas into their tools and are merely tinkering with the user friendliness of their systems and improving their input and output capabilities (for example, report generation). The only real development of late has been to upgrade these packages to Windows™ based products.

7.8 EVALUATION

7.8.1 The Software Development Industry

In order to avoid developing theories within the closeted walls of academia, contact was made with a number of companies throughout the development of this thesis. Thanks must go to these companies who provided valuable insights into the 'coal face' of software development in the real world. These contacts were used to assess the popularity of the concepts developed and to keep in touch with real world approaches to project management used in the development of software. During the development of this thesis a questionnaire was used (Appendix H) to determine a more objective view of what was happening in the real world. This was backed up with visits to companies to discuss ideas with project managers experiencing real risks and uncertainties with their own software. Questionnaires must always be treated with some caution as the information they provide may be biased depending on how questions are raised. Both qualitative and quantitative information was provided through the feedback from this source. Due to the limited number of local companies available for interview (twelve in all) qualitative analysis proved more valuable due to the nature of the information sought. A broad cross section of companies were interviewed ranging from large nationals (for example, BT and GPT) to small, local software houses.

The response rate for the questionnaire was pleasantly high with 60% of those companies contacted responding. The companies that did reply provided quite positive information with regard to the ideas developed. This questionnaire also provided a means of assessing each of the companies approaches to the development of software. The general feeling was that software houses are, on the whole, a number of years

behind the concepts and ideas been produced within academia.

From the companies who responded, one third (four companies) used no identifiable software development model or methodology. The excuse in these cases was down to the size of development projects undertaken that required no more than one person-month of effort. These projects were, therefore, performed using an early build-and-fix model without any real specification or requirements analysis. A specification of sorts would be provided by a client, and a programmer would be assigned the task of putting together a solution. One questions, in these cases, how organisational objectives were reached. How are these companies managing long term objectives when they appear to be working to only short term goals at any one time?

All of the other companies interviewed had an identifiable software development strategy and employed particular methods for developing their systems. For example, both GPT and BT use Cascade®, (a propriety tool and methodology) that not only provides them with an organisation-wide project management tool, but also with a methodology with which all departments can work. Other companies used methods such as SSADM, E-R Modelling, and Formal Methods, and one technique adopted was to use reusable code (one company developing its systems from 80% reusable code). The traditional life cycle approach was also identified as a model used by several of the companies interviewed (30% for this case).

None of the companies felt that project management has been used within their organisation optimally. Both BT at Martlesham Heath and GPT at Nottingham had done something about this, and introduced an organisation-wide methodology based on Cascade®. The other companies felt there was something lacking with the approaches being used, and identified that some of the developments made within this thesis would be of practical benefit to them. They highlighted identification of project risks, and the ability to plan for alternatives, as the most beneficial parts of a new system. Modelling different time and cost functions was also identified as a need although, as has been noted in previous chapters, there are several project management packages that already offer this facility.

More powerful planning techniques were also needed by two of the companies interviewed. These companies identified that their current practices were not flexible enough and felt that Generalised Activity Networks would provide a more visual representation of their projects.

The most likely outcome of a project is clearly important to many managers and this was identified as a result lacking in most approaches. Modal estimates provide this

result. One company was also interested in having an intelligent project management tool that could try various *what if* analysis to provide several possible project development scenarios. A smaller company was also interested in these ideas especially if larger defence contracts were undertaken. More management reporting facilities were also identified as a need and this was backed up from discussions at Project Management South (1994).

An overall impression that one had after speaking with several companies was the large split between academia and industry. It was felt that many companies, although strong in their own particular fields (for example, telecommunications, health care, general consultancy), had software development concepts that were still based on 1960s' ideas. It is obviously going to be some time before more dynamic approaches are widely accepted in the software development industry.

7.8.2 An Example of a Project With Little Requirement for Generalised Activity Networks

Unfortunately, applying probabilistic rules to one or two projects can never prove to be a fair assessment of Generalised Activity Networks. At the end of the day all projects are seen to complete, and are signed off, in a particular time, as are individual activities within those projects (although this completion can be somewhat difficult to define in software development projects). One would argue with hindsight, having planned a project and seen it through to completion, that that project could have easily have been modelled by deterministic means. Unfortunately, this does not help the evaluation of probabilistic techniques.

Generalised Activity Networks provide a means of explicitly identifying potential risk points in a project. Identifying these risks is the first step in performing risk management. By forcing managers to plan risks they become aware of the consequences and can reduce these risks (either by *avoidance*, *deflection* or *contingencies* [Turner 1993]). Having explicitly identified a project's possibilities and its risks within a Generalised Activity Network, its analysis provides details of the most likely outcome of that project (ie the mode rather than the mean). The detailed analysis of such plans however, must be questioned as 'these systems [Monte Carlo, PERT] must be treated with extreme care or the analysis takes over. You will spend an inordinate amount of time producing data of little value' [Turner 1993]. This point was addressed in chapter four as this analysis was improved to ensure that the data produced was of some value and the time to produce this data was minimised.

The first example project (presented in Appendix A) shows that there are projects in

which risks are limited. Standard engineering projects usually fall into this category - activities have been performed many times before and experienced managers are usually aware of all the possible risks.

The project outlined in Appendix A provides a good example of how useful current project management information systems (network based) are in these engineering-type projects. These plans represent the Milltown Bridge Project - an engineering project with little risk or variation. Page A1 in the appendices shows a detailed Work Breakdown Structure of this project from the integrative level down. The tactical level of the project is derived from the strategic level that shows simply *preconstruction, construction* and *finals*. Page A2 shows an Activity-on-the-Node representation of the project (based at a more strategic level) and pages A3 to A8 show various project Gantt charts. These charts provide a detailed representation of the project in a report form that is easily understood by all levels of the project's team. They also provide important managerial reports that are used for project tendering, and hopefully securing a contract in the first place. For projects of this nature (ie reasonably deterministic) plans of this form are invaluable.

In these situations Generalised Activity Networks are perhaps unnecessary as plans are based on activities that have been performed many times before and have little risk associated with them. However, should there be uncertainties in a project, these could be modelled explicitly by Generalised Activity Networks. The advantage of Generalised Activity Networks in this case is that they still allow projects to be planned in the same straight forward way as that shown in Appendix A. They are powerful enough to provide the flexibility as and when it is required.

This example shows that for general engineering-type projects the powerful facilities offered by Generalised Activity Networks are, in many cases, unnecessary. As most software developments are based on these established engineering practices it is not surprising that they have adopted the same deterministic managerial principles. Unfortunately, software developments (along with other project types - for example, research and development projects) have already been shown to be much more dynamic in nature. It is these dynamic projects that require more flexible management support systems such as Generalised Activity Networks. This will be born out in the following example.

7.8.3 An Example Project Where Metamodels and Generalised Activity Networks Give Benefits

In order to evaluate Generalised Activity Networks and metamodels, it is necessary to

look at a software project where problems were encountered. It is important to see how those problems were overcome by the project manager involved and how the concepts developed in this thesis would have helped. It was not necessary to search very far for a project that fitted this requirement as the first software project studied provided the required results. The project chosen for this study is a small, eleven person-month software development undertaken at Derby (larger projects have already been documented in other papers - for example Kidd (1990)). The intention of this project was to create an educational software package for use in primary schools. The package was completed in a collaboration between the University of Derby and Redhill Primary School, Nottingham.

The initial approach to this project was to use the traditional software development life cycle with which most companies are familiar. This is represented in figure 7.3 which shows the initial strategic level plan (Activity-on-the-Node in this case) drawn up during the first week of the project. The project was observed from a distance - in other words no affect was imposed on the development of this project - its development was merely monitored.

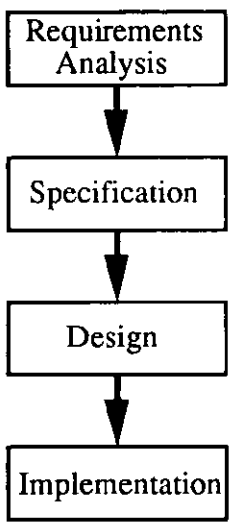


Figure 7.3 The Initial Project Plan

Figure 7.3 actually represents a standard waterfall approach along the lines of Analysis, Specification, Design and Implementation. When interviewing the project manager at the end of this project he stated 'In retrospect it was impossible to decide on these issues at that point in the project. This was because a greater understanding of the problems associated with providing educational software was required'.

It was not until two months into the project that the traditional development approach was found to be totally inadequate for a project of this nature. While the specification was been put together it became clear that the project was way off course. Teachers were uncomfortable about been involved in the software development process as they are used to ready-made products. This suggested using prototypes so that the teachers could be prompted into providing feedback. This prototype development, therefore, needed to be introduced into the project plan.

When the project hit these problems, the project plan was radically altered to incorporate prototyping. It was followed to the successful completion of the project. The actual path this project took is shown in figure 7.4. This strategic level plan is taken directly from the metamodel developed in chapter two. Had this metamodel been available to the project manager from the outset, the use of prototypes could have been adopted more smoothly. The final plan, in the words of the manager, was a hybrid of several models. This emphasises the uses of combined models or metamodels that are clearly of benefit to many software development projects.

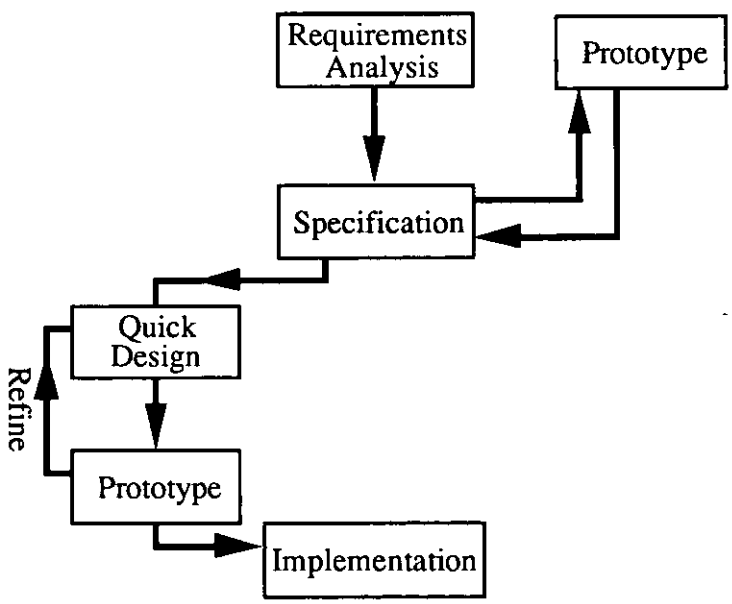


Figure 7.4 Final Project Plan

A question one could ask is why a prototyping model was not adopted from the outset. This model would have provided an adequate framework in which this project could have evolved. Unfortunately, this project emphasises the main difficulty of software - the problem domain and project environment is never fully understood at the outset. Figure 7.5 goes some way to emphasising this point. This figure shows the

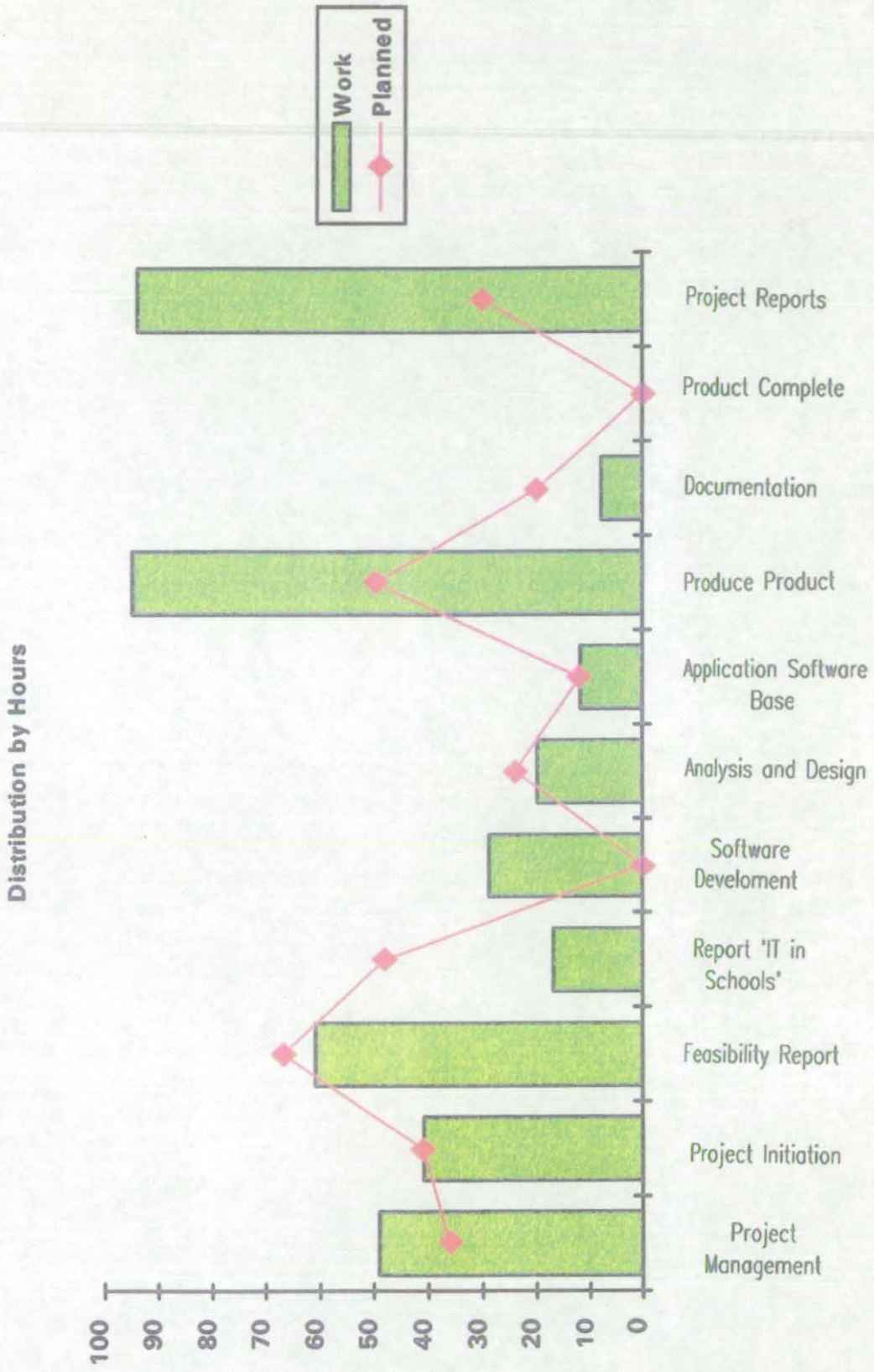


Figure 7.5 Difference Between Planned and Performed Activity Hours

considerable differences between planned and performed activities in this project. The uncertainties inherent within software developments lead to this unpredictable performance. Had it been known that prototypes were required it is true to say that a prototyping model could have been used. However, it was not until the project was planned and underway that this requirement was determined. It is only when the software development project is undertaken that its true environment comes to light.

The metamodel copes with this problem by identifying all possible alternatives at different stages of a project's progression. By combining several models within one metamodel, these approaches can be adopted as and when they are required. This allows a manager to adopt and adapt alternative development models at applicable stages in a project life cycle. Managers are not restricted to one model and can accommodate a dynamic systems development without major replanning.

In retrospect one could have put together the final project plan (figure 7.4) at the outset using a Deterministic Activity Network in a popular project management tool (although the loop would have had to be coded implicitly). This was obviously impossible as any probabilistic branching, that was unknown at the initial stages of this project development, could not have been identified within such a plan. The plan put together at the end of the project is not a genuine reflection of the problem domain at the outset.

In conclusion, by encountering several problems at its initial stages, this project provided a valuable insight into real software project difficulties. The metamodel would have provided this project manager with a more flexible model with which to work. Frantic replanning after two months could have been avoided as variations could have been accommodated by a metamodel plan. The introduction of a prototyping approach could have been adopted more smoothly as the project plan would not have needed changing. The plan would merely have been used to identify the alternative strategic route pursued and would have provided an acceptable means of project control. A Generalised Activity Network representing this project would also have helped to control the development process by allowing the project manager to explicitly identify alternative approaches within the plan. Although estimates of probabilistic branching may have been sketchy at the initial stages (for example, what would have been the assigned probability for the prototype approach?), any alternatives would have already been brought to the attention of the project manager. It would only be through experience, like all other project estimation, that accurate project estimates can be made. It is worth noting that the estimates of probabilistic branching within a project plan would improve as the project progressed. These estimates would ultimately reach certainty when the branch point was reached.

7.9 CONCLUSION

7.9.1 Contribution

This thesis has made a significant contribution to three areas of software development and project management. First, the concept of hybrid models or metamodels has been developed to assist a more dynamic development of software systems. Second, a project management technique (activity networks) has been improved and developed and made more applicable to industrial requirements today. Third, current analysis techniques that use simulation for activity networks have been improved in terms of both efficiency and quality of data identified (backed up by experimental results). These three contributions have shown that a more flexible planning approach required for software development can and should be adopted.

7.9.2 Summary

Overall this thesis has made substantial progress in several areas involved with the management of the software development process. Initial aims and objectives have been reached and taken further and many new avenues of research have been opened up.

In conclusion, even after thirty years, the field of software development management is in its infancy. It has a long way to go before the software crisis is averted. Only by adopting more flexible, dynamic approaches to software development can this problem be elevated. The advances made in this thesis go some way to allowing these approaches to be adopted.

Appendices

APPENDIX A

An Example Project

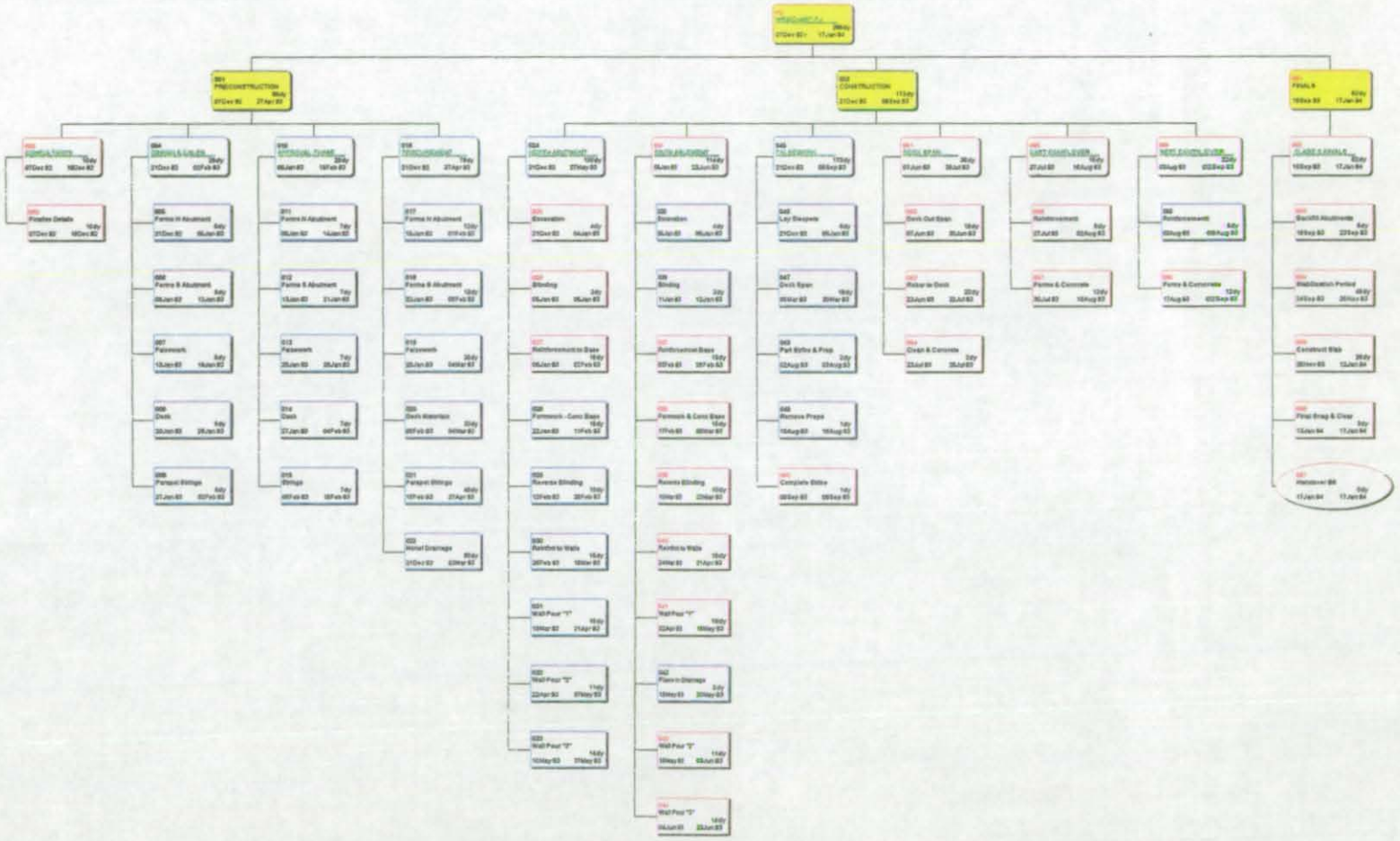
A.1 OVERVIEW

This appendix presents project documentation associated with the Milltown Road Bridge Project. This project began in December, 1992 and was completed successfully in January, 1994. The extracts are listings produced by CA-SuperProject® version 3 'C'. Thanks must go to Computer Associates® and TBV Consult for allowing these project plans to be reproduced here.

A.2 STAFF

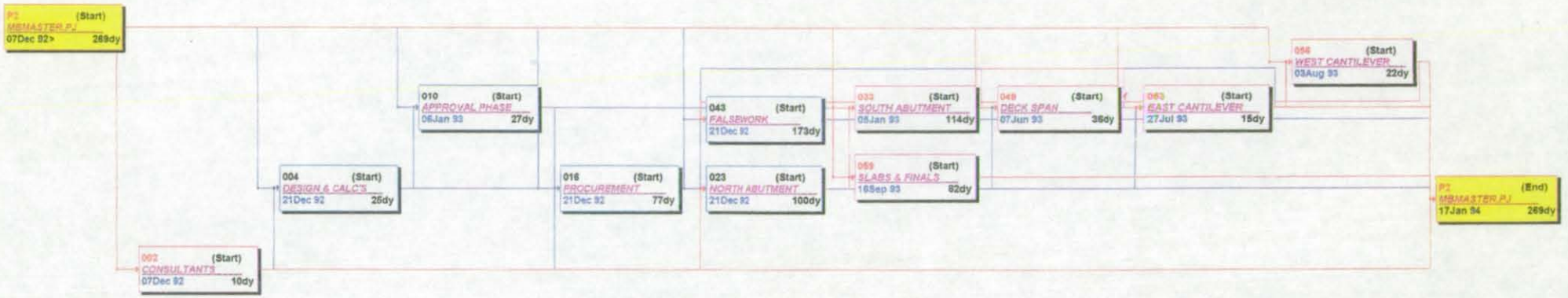
Consultant Engineer:	Murray Walker
Main Contractor:	Scott Bradley
Sub Contractor:	J Fitzgerald
Programme Managers:	Mike Harvey (TBV Consult, Croydon)
	Jonathan Reece (TBV Consult, Croydon)
Client:	Department of Transport

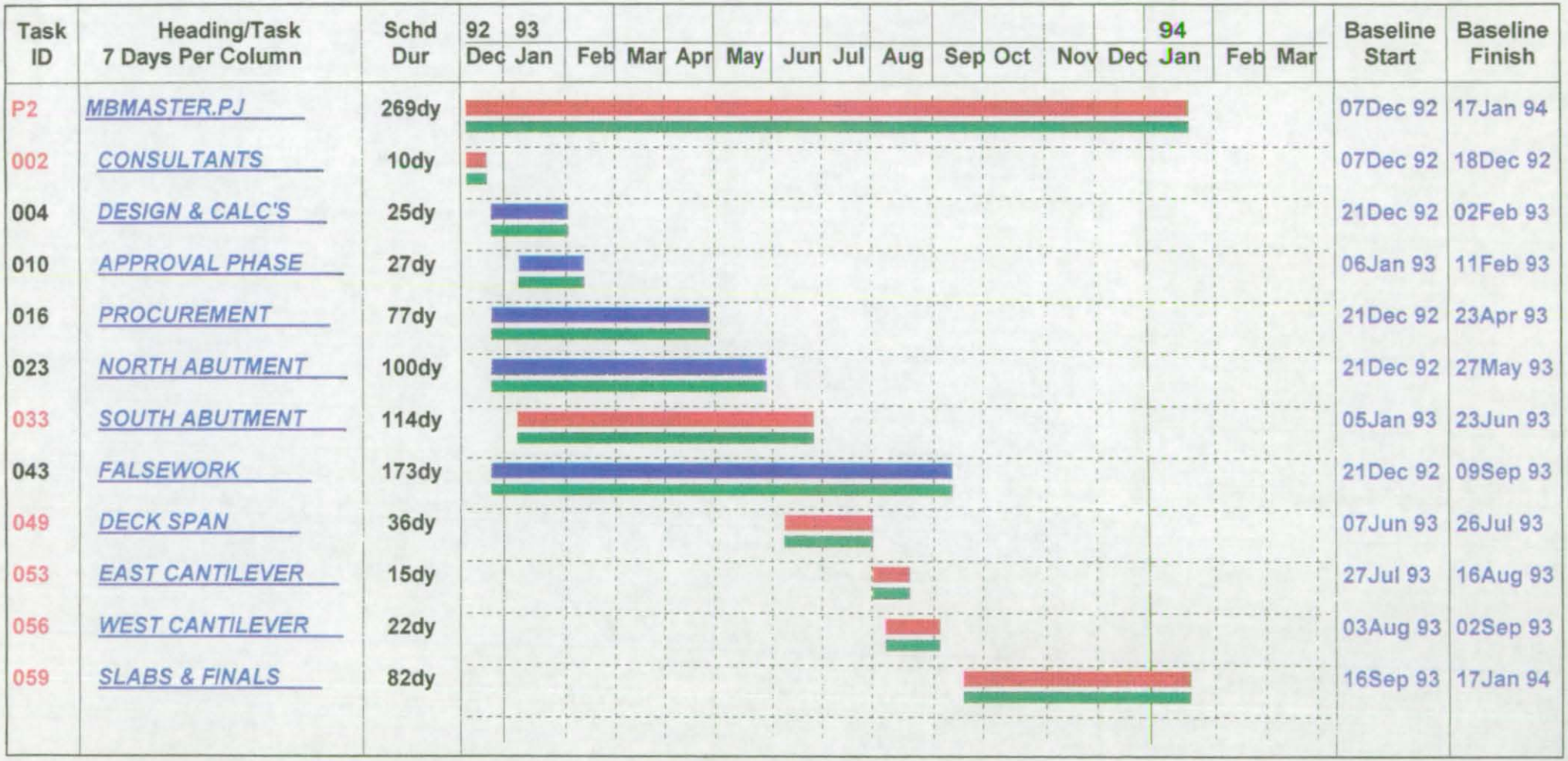
A.3 LISTINGS

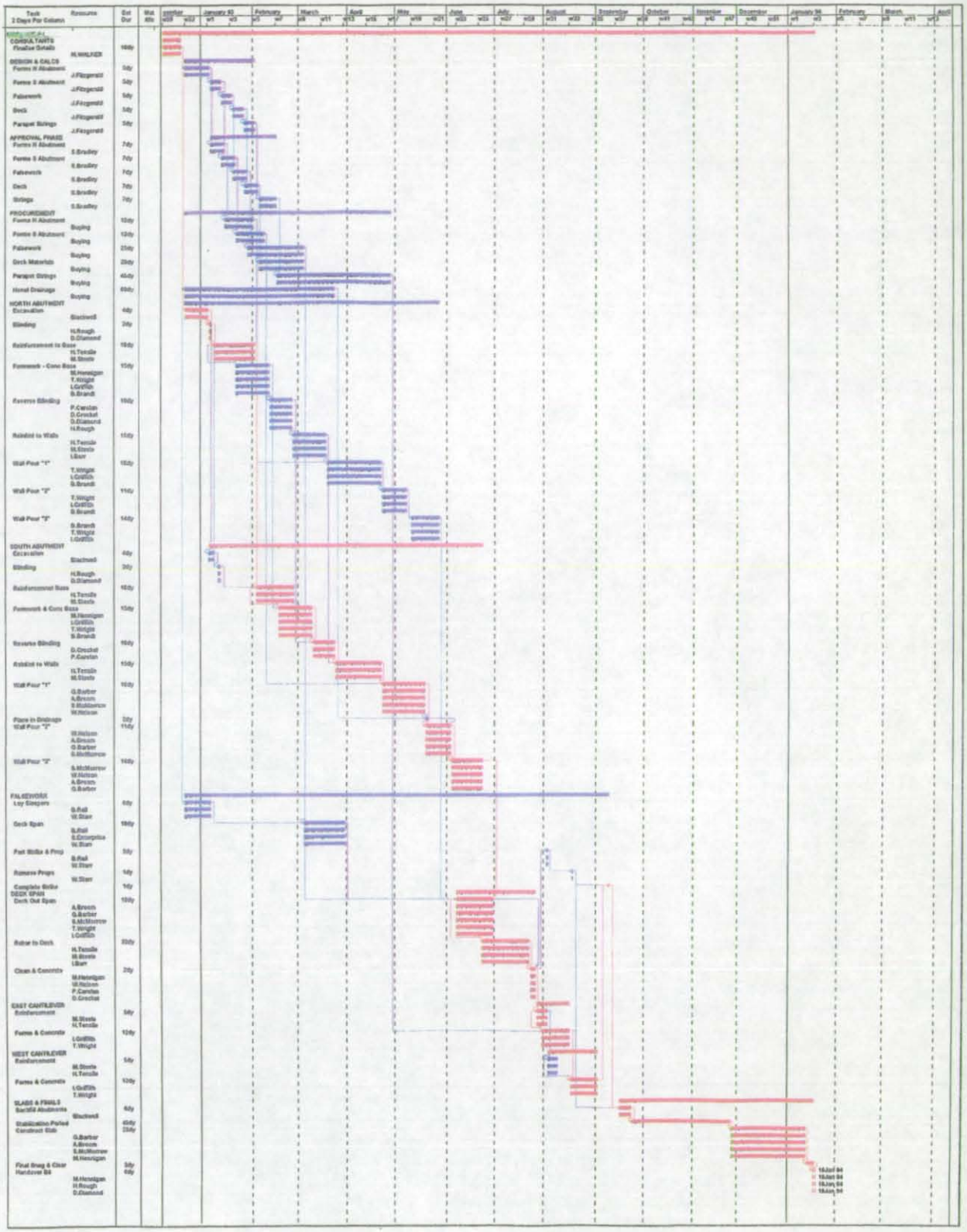


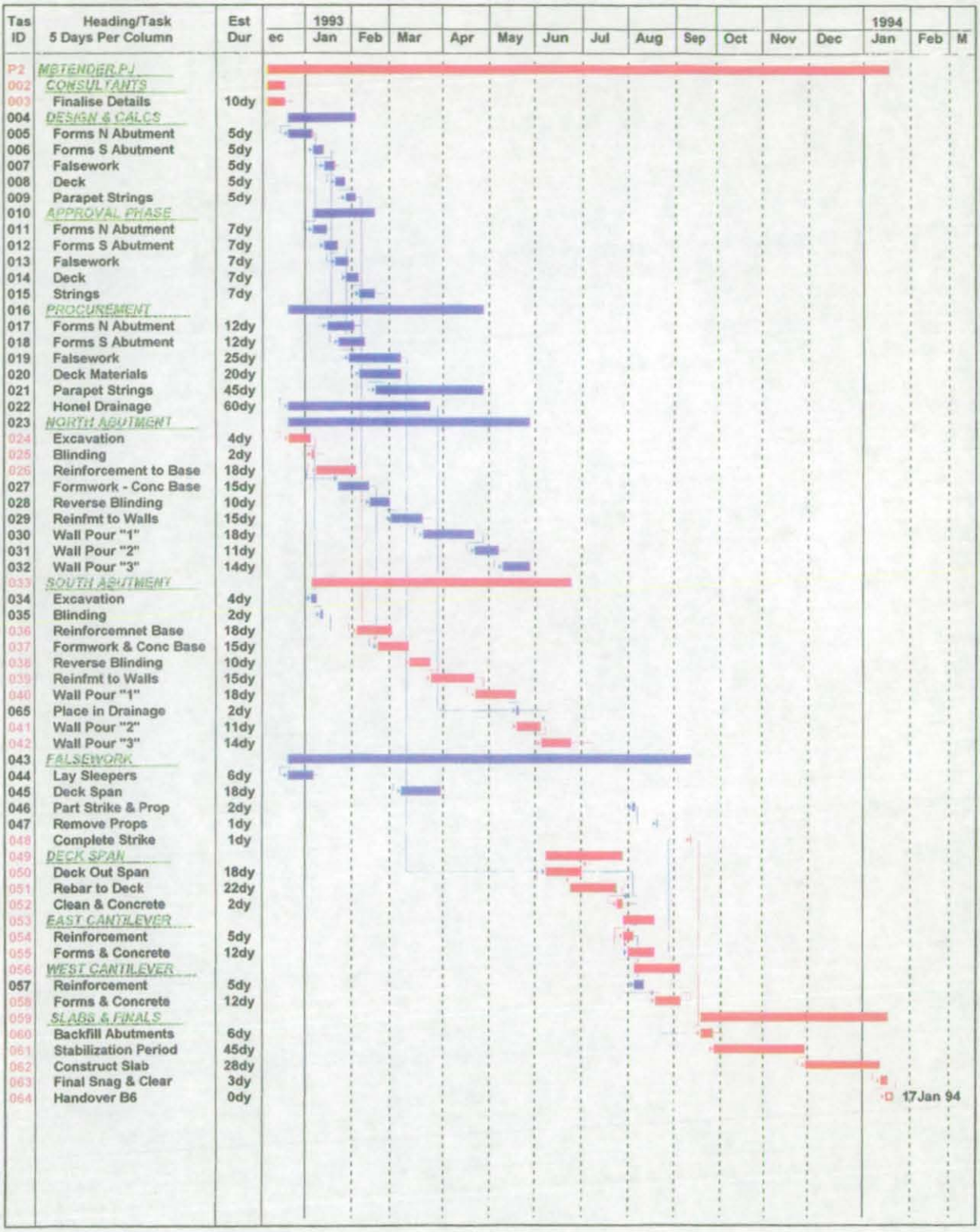
A1

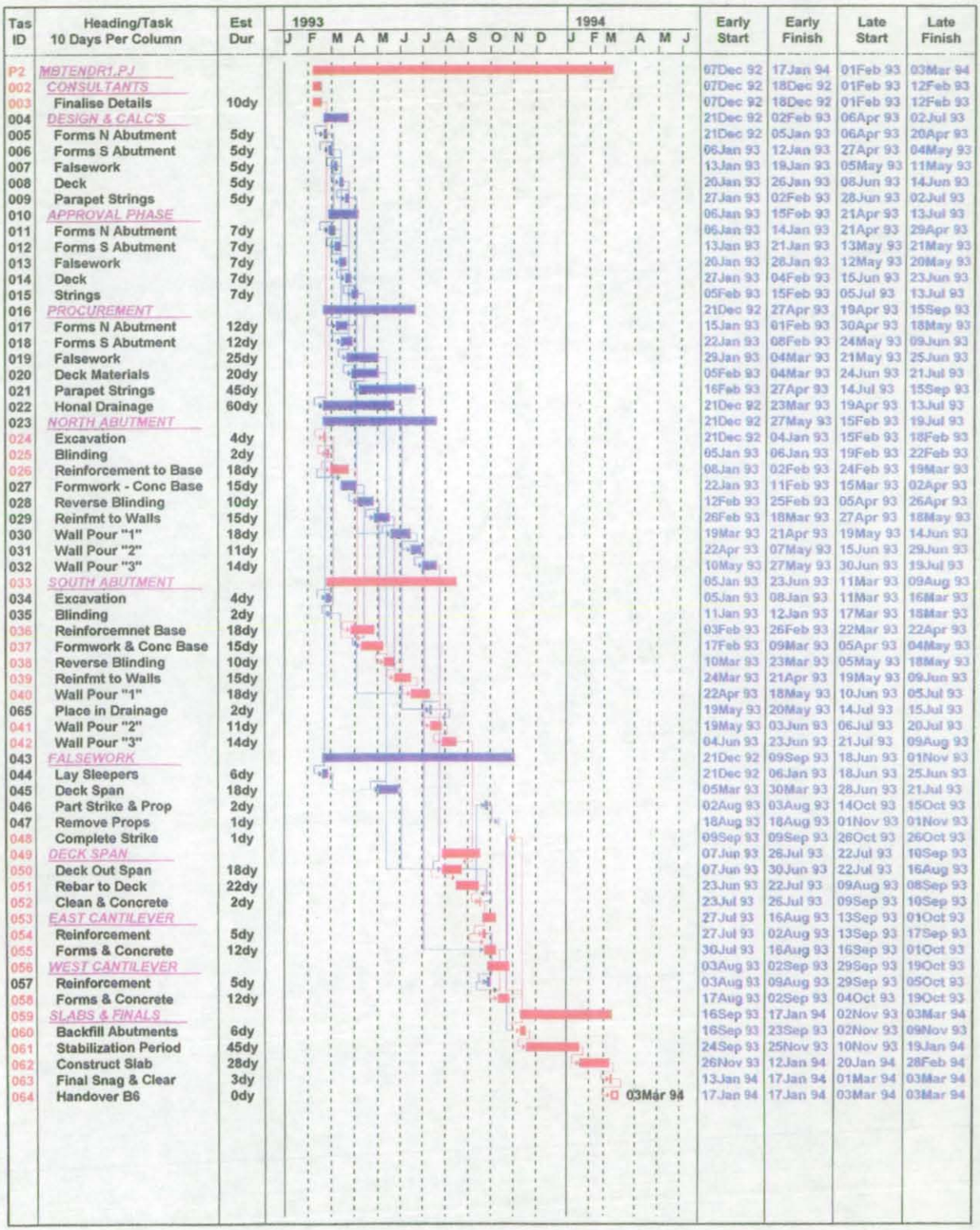
A2


P2 (Start)
MBMASTER.PJ
07Dec 92> 269dy
002 (Start)
CONSULTANTS
07Dec 92 10dy
004 (Start)
DESIGN & CALC'S
21Dec 92 26dy
010 (Start)
APPROVAL PHASE
06Jan 93 27dy
016 (Start)
PROCUREMENT
21Dec 92 77dy
023 (Start)
NORTH ABUTMENT
21Dec 92 100dy
043 (Start)
FALSEWORK
21Dec 92 173dy
032 (Start)
SOUTH ABUTMENT
05Jan 93 114dy
059 (Start)
SLABS & FINALS
16Sep 93 82dy
049 (Start)
DECK SPAN
07Jun 93 36dy
063 (Start)
EAST CANTILEVER
27Jul 93 15dy
056 (Start)
WEST CANTILEVER
03Aug 93 22dy
P2 (End)
MBMASTER.PJ
17Jan 94 269dy

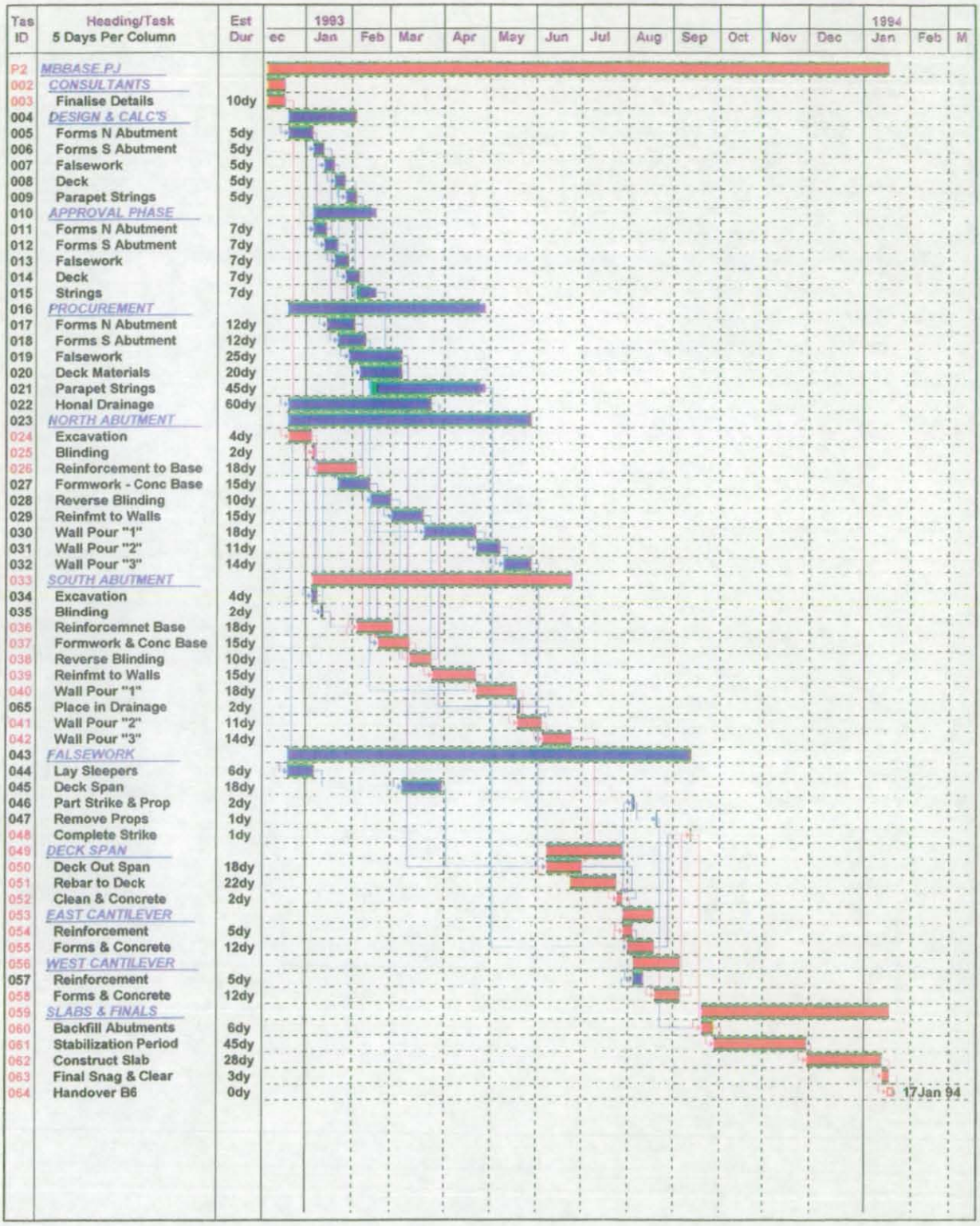

Task ID
Heading/Task 7 Days Per Column
Schd Dur
92
93
94
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
Mar
Baseline Start
Baseline Finish
P2 MBMASTER.PJ 269dy 07Dec 92 17Jan 94
002 CONSULTANTS 10dy 07Dec 92 18Dec 92
004 DESIGN & CALC'S 25dy 21Dec 92 02Feb 93
010 APPROVAL PHASE 27dy 06Jan 93 11Feb 93
016 PROCUREMENT 77dy 21Dec 92 23Apr 93
023 NORTH ABUTMENT 100dy 21Dec 92 27May 93
033 SOUTH ABUTMENT 114dy 05Jan 93 23Jun 93
043 FALSEWORK 173dy 21Dec 92 09Sep 93
049 DECK SPAN 36dy 07Jun 93 26Jul 93
053 EAST CANTILEVER 15dy 27Jul 93 16Aug 93
056 WEST CANTILEVER 22dy 03Aug 93 02Sep 93
059 SLABS & FINALS 82dy 16Sep 93 17Jan 94

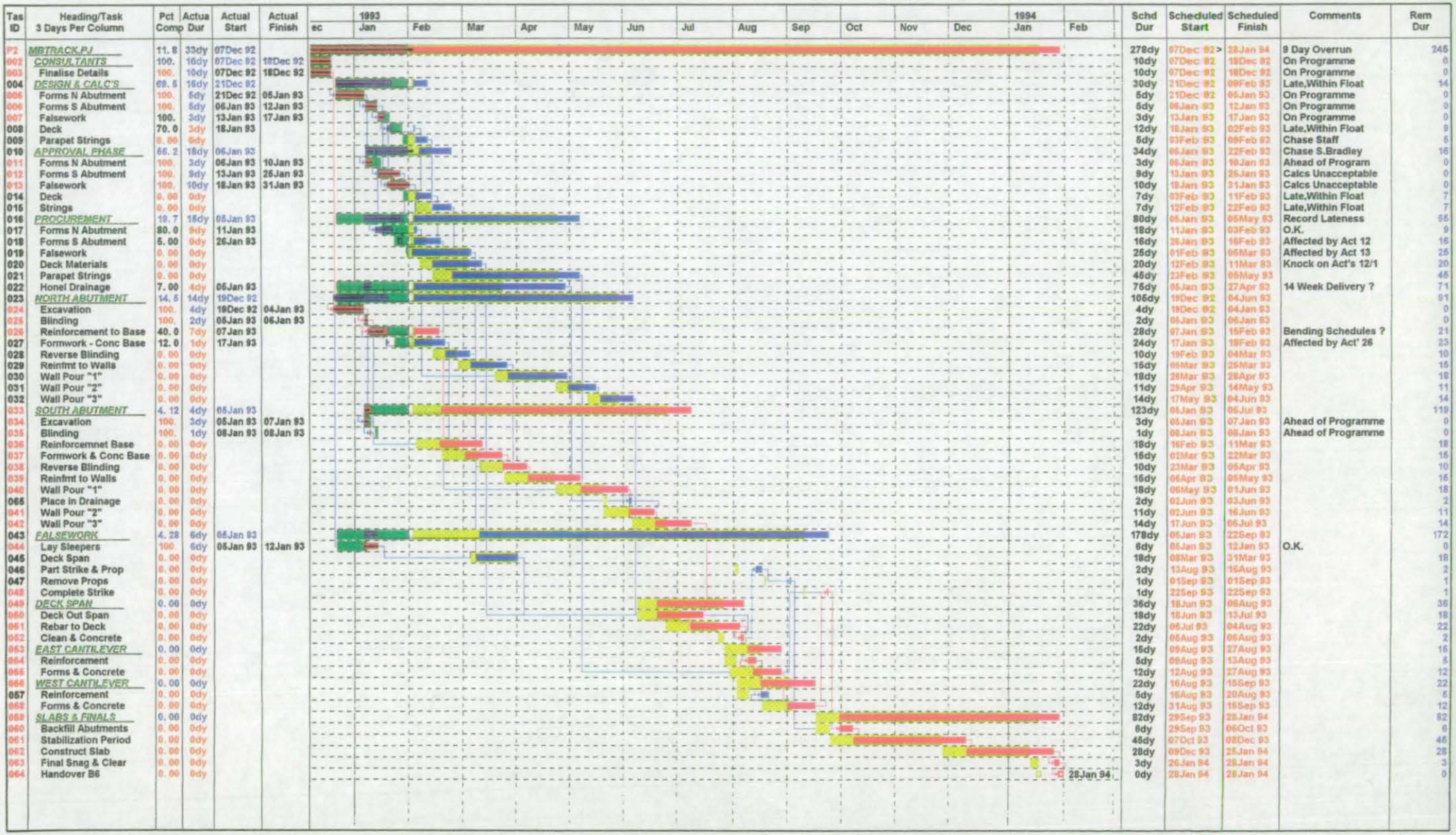



Tas ID
Heading/Task 5 Days Per Column
Est Dur
1993
1994
ec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
M
P2 METENDER PJ
002 CONSULTANTS
003 Finalise Details 10dy
004 DESIGN & CALCS
005 Forms N Abutment 5dy
006 Forms S Abutment 5dy
007 Falsework 5dy
008 Deck 5dy
009 Parapet Strings 5dy
010 APPROVAL PHASE
011 Forms N Abutment 7dy
012 Forms S Abutment 7dy
013 Falsework 7dy
014 Deck 7dy
015 Strings 7dy
016 PROCUREMENT
017 Forms N Abutment 12dy
018 Forms S Abutment 12dy
019 Falsework 25dy
020 Deck Materials 20dy
021 Parapet Strings 45dy
022 Honel Drainage 60dy
023 NORTH ABUTMENT
024 Excavation 4dy
025 Blinding 2dy
026 Reinforcement to Base 18dy
027 Formwork - Conc Base 15dy
028 Reverse Blinding 10dy
029 Reinfmt to Walls 15dy
030 Wall Pour "1" 18dy
031 Wall Pour "2" 11dy
032 Wall Pour "3" 14dy
033 SOUTH ABUTMENT
034 Excavation 4dy
035 Blinding 2dy
036 Reinforcemnet Base 18dy
037 Formwork & Conc Base 15dy
038 Reverse Blinding 10dy
039 Reinfmt to Walls 15dy
040 Wall Pour "1" 18dy
065 Place in Drainage 2dy
041 Wall Pour "2" 11dy
042 Wall Pour "3" 14dy
043 FALSEWORK
044 Lay Sleepers 6dy
045 Deck Span 18dy
046 Part Strike & Prop 2dy
047 Remove Props 1dy
048 Complete Strike 1dy
049 DECK SPAN
050 Deck Out Span 18dy
051 Rebar to Deck 22dy
052 Clean & Concrete 2dy
053 EAST CANTILEVER
054 Reinforcement 5dy
055 Forms & Concrete 12dy
056 WEST CANTILEVER
057 Reinforcement 5dy
058 Forms & Concrete 12dy
059 SLABS & FINALS
060 Backfill Abutments 6dy
061 Stabilization Period 45dy
062 Construct Slab 28dy
063 Final Snag & Clear 3dy
064 Handover B6 0dy
17Jan 94



Tas ID	Heading/Task 10 Days Per Column	Est Dur	Early Start	Early Finish	Late Start	Late Finish
P2	MBTENDR1.PJ		07Dec 92	17Jan 94	01Feb 93	03Mar 94
002	CONSULTANTS		07Dec 92	18Dec 92	01Feb 93	12Feb 93
003	Finalise Details	10dy	07Dec 92	18Dec 92	01Feb 93	12Feb 93
004	DESIGN & CALC'S		21Dec 92	02Feb 93	06Apr 93	02Jul 93
005	Forms N Abutment	5dy	21Dec 92	05Jan 93	06Apr 93	20Apr 93
006	Forms S Abutment	5dy	06Jan 93	12Jan 93	27Apr 93	04May 93
007	Falsework	5dy	13Jan 93	19Jan 93	05May 93	11May 93
008	Deck	5dy	20Jan 93	26Jan 93	08Jun 93	14Jun 93
009	Parapet Strings	5dy	27Jan 93	02Feb 93	28Jun 93	02Jul 93
010	APPROVAL PHASE		06Jan 93	15Feb 93	21Apr 93	13Jul 93
011	Forms N Abutment	7dy	06Jan 93	14Jan 93	21Apr 93	29Apr 93
012	Forms S Abutment	7dy	13Jan 93	21Jan 93	13May 93	21May 93
013	Falsework	7dy	20Jan 93	28Jan 93	12May 93	20May 93
014	Deck	7dy	27Jan 93	04Feb 93	15Jun 93	23Jun 93
015	Strings	7dy	05Feb 93	15Feb 93	05Jul 93	13Jul 93
016	PROCUREMENT		21Dec 92	27Apr 93	19Apr 93	15Sep 93
017	Forms N Abutment	12dy	15Jan 93	01Feb 93	30Apr 93	18May 93
018	Forms S Abutment	12dy	22Jan 93	08Feb 93	24May 93	09Jun 93
019	Falsework	25dy	29Jan 93	04Mar 93	21May 93	25Jun 93
020	Deck Materials	20dy	05Feb 93	04Mar 93	24Jun 93	21Jul 93
021	Parapet Strings	45dy	16Feb 93	27Apr 93	14Jul 93	15Sep 93
022	Honal Drainage	60dy	21Dec 92	23Mar 93	19Apr 93	13Jul 93
023	NORTH ABUTMENT		21Dec 92	27May 93	15Feb 93	19Jul 93
024	Excavation	4dy	21Dec 92	04Jan 93	15Feb 93	18Feb 93
025	Blinding	2dy	05Jan 93	06Jan 93	19Feb 93	22Feb 93
026	Reinforcement to Base	18dy	08Jan 93	02Feb 93	24Feb 93	19Mar 93
027	Formwork - Conc Base	15dy	22Jan 93	11Feb 93	15Mar 93	02Apr 93
028	Reverse Blinding	10dy	12Feb 93	25Feb 93	05Apr 93	26Apr 93
029	Reinfmt to Walls	15dy	26Feb 93	18Mar 93	27Apr 93	18May 93
030	Wall Pour "1"	18dy	19Mar 93	21Apr 93	19May 93	14Jun 93
031	Wall Pour "2"	11dy	22Apr 93	07May 93	15Jun 93	29Jun 93
032	Wall Pour "3"	14dy	10May 93	27May 93	30Jun 93	19Jul 93
033	SOUTH ABUTMENT		05Jan 93	23Jun 93	11Mar 93	09Aug 93
034	Excavation	4dy	05Jan 93	08Jan 93	11Mar 93	16Mar 93
035	Blinding	2dy	11Jan 93	12Jan 93	17Mar 93	18Mar 93
036	Reinforcemnet Base	18dy	03Feb 93	26Feb 93	22Mar 93	22Apr 93
037	Formwork & Conc Base	15dy	17Feb 93	09Mar 93	05Apr 93	04May 93
038	Reverse Blinding	10dy	10Mar 93	23Mar 93	05May 93	18May 93
039	Reinfmt to Walls	15dy	24Mar 93	21Apr 93	19May 93	09Jun 93
040	Wall Pour "1"	18dy	22Apr 93	18May 93	10Jun 93	05Jul 93
065	Place in Drainage	2dy	19May 93	20May 93	14Jul 93	15Jul 93
041	Wall Pour "2"	11dy	19May 93	03Jun 93	06Jul 93	20Jul 93
042	Wall Pour "3"	14dy	04Jun 93	23Jun 93	21Jul 93	09Aug 93
043	FALSEWORK		21Dec 92	09Sep 93	18Jun 93	01Nov 93
044	Lay Sleepers	6dy	21Dec 92	06Jan 93	18Jun 93	25Jun 93
045	Deck Span	18dy	05Mar 93	30Mar 93	28Jun 93	21Jul 93
046	Part Strike & Prop	2dy	02Aug 93	03Aug 93	14Oct 93	15Oct 93
047	Remove Props	1dy	16Aug 93	16Aug 93	01Nov 93	01Nov 93
048	Complete Strike	1dy	09Sep 93	09Sep 93	26Oct 93	26Oct 93
049	DECK SPAN		07Jun 93	26Jul 93	22Jul 93	10Sep 93
050	Deck Out Span	18dy	07Jun 93	30Jun 93	22Jul 93	16Aug 93
051	Rebar to Deck	22dy	23Jun 93	22Jul 93	09Aug 93	08Sep 93
052	Clean & Concrete	2dy	23Jul 93	26Jul 93	09Sep 93	10Sep 93
053	EAST CANTILEVER		27Jul 93	16Aug 93	13Sep 93	01Oct 93
054	Reinforcement	5dy	27Jul 93	02Aug 93	13Sep 93	17Sep 93
055	Forms & Concrete	12dy	30Jul 93	16Aug 93	16Sep 93	01Oct 93
056	WEST CANTILEVER		03Aug 93	02Sep 93	29Sep 93	19Oct 93
057	Reinforcement	5dy	03Aug 93	09Aug 93	29Sep 93	05Oct 93
058	Forms & Concrete	12dy	17Aug 93	02Sep 93	04Oct 93	19Oct 93
059	SLABS & FINALS		16Sep 93	17Jan 94	02Nov 93	03Mar 94
060	Backfill Abutments	6dy	16Sep 93	23Sep 93	02Nov 93	09Nov 93
061	Stabilization Period	45dy	24Sep 93	25Nov 93	10Nov 93	19Jan 94
062	Construct Slab	28dy	26Nov 93	12Jan 94	20Jan 94	28Feb 94
063	Final Snag & Clear	3dy	13Jan 94	17Jan 94	01Mar 94	03Mar 94
064	Handover B6	0dy	17Jan 94	17Jan 94	03Mar 94	03Mar 94


Tas ID
Heading/Task
5 Days Per Column
Est Dur
1993
1994
ec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
M
P2 MBBASE PJ
002 CONSULTANTS
003 Finalise Details 10dy
004 DESIGN & CALC'S
005 Forms N Abutment 5dy
006 Forms S Abutment 5dy
007 Falsework 5dy
008 Deck 5dy
009 Parapet Strings 5dy
010 APPROVAL PHASE
011 Forms N Abutment 7dy
012 Forms S Abutment 7dy
013 Falsework 7dy
014 Deck 7dy
015 Strings 7dy
016 PROCUREMENT
017 Forms N Abutment 12dy
018 Forms S Abutment 12dy
019 Falsework 25dy
020 Deck Materials 20dy
021 Parapet Strings 45dy
022 Honal Drainage 60dy
023 NORTH ABUTMENT
024 Excavation 4dy
025 Blinding 2dy
026 Reinforcement to Base 18dy
027 Formwork - Conc Base 15dy
028 Reverse Blinding 10dy
029 Reinfmt to Walls 15dy
030 Wall Pour "1" 18dy
031 Wall Pour "2" 11dy
032 Wall Pour "3" 14dy
033 SOUTH ABUTMENT
034 Excavation 4dy
035 Blinding 2dy
036 Reinforcemnet Base 18dy
037 Formwork & Conc Base 15dy
038 Reverse Blinding 10dy
039 Reinfmt to Walls 15dy
040 Wall Pour "1" 18dy
065 Place in Drainage 2dy
041 Wall Pour "2" 11dy
042 Wall Pour "3" 14dy
043 FALSEWORK
044 Lay Sleepers 6dy
045 Deck Span 18dy
046 Part Strike & Prop 2dy
047 Remove Props 1dy
048 Complete Strike 1dy
049 DECK SPAN
050 Deck Out Span 18dy
051 Rebar to Deck 22dy
052 Clean & Concrete 2dy
053 EAST CANTILEVER
054 Reinforcement 5dy
055 Forms & Concrete 12dy
056 WEST CANTILEVER
057 Reinforcement 5dy
058 Forms & Concrete 12dy
059 SLABS & FINALS
060 Backfill Abutments 6dy
061 Stabilization Period 45dy
062 Construct Slab 28dy
063 Final Snag & Clear 3dy
064 Handover B6 0dy
17Jan 94



Tas ID	Heading/Task 3 Days Per Column	Pct Comp	Actua Dur	Actual Start	Actual Finish	Schd Dur	Scheduled Start	Scheduled Finish	Comments	Rem Dur
P2	MBTRACK.PJ	11.8	33dy	07Dec 92		278dy	07Dec 92 >	28Jan 94	9 Day Overrun	245
002	CONSULTANTS	100.	10dy	07Dec 92	18Dec 92	10dy	07Dec 92	18Dec 92	On Programme	0
003	Finalise Details	100.	10dy	07Dec 92	18Dec 92	10dy	07Dec 92	18Dec 92	On Programme	0
004	DESIGN & CALC'S	69.5	16dy	21Dec 92		30dy	21Dec 92	09Feb 93	Late,Within Float	14
006	Forms N Abutment	100.	6dy	21Dec 92	05Jan 93	5dy	21Dec 92	05Jan 93	On Programme	0
006	Forms S Abutment	100.	6dy	06Jan 93	12Jan 93	5dy	06Jan 93	12Jan 93	On Programme	0
007	Falsework	100.	3dy	13Jan 93	17Jan 93	3dy	13Jan 93	17Jan 93	On Programme	0
008	Deck	70.0	3dy	18Jan 93		12dy	18Jan 93	02Feb 93	Late,Within Float	9
009	Parapet Strings	0.00	0dy			5dy	03Feb 93	09Feb 93	Chase Staff	5
010	APPROVAL PHASE	66.2	18dy	06Jan 93		34dy	06Jan 93	22Feb 93	Chase S.Bradley	16
011	Forms N Abutment	100.	3dy	06Jan 93	10Jan 93	3dy	06Jan 93	10Jan 93	Ahead of Program	0
012	Forms S Abutment	100.	9dy	13Jan 93	26Jan 93	9dy	13Jan 93	26Jan 93	Calcs Unacceptable	0
013	Falsework	100.	10dy	18Jan 93	31Jan 93	10dy	18Jan 93	31Jan 93	Calcs Unacceptable	0
014	Deck	0.00	0dy			7dy	03Feb 93	11Feb 93	Late,Within Float	7
015	Strings	0.00	0dy			7dy	12Feb 93	22Feb 93	Late,Within Float	7
016	PROCUREMENT	19.7	18dy	06Jan 93		80dy	06Jan 93	05May 93	Record Lateness	65
017	Forms N Abutment	80.0	9dy	11Jan 93		18dy	11Jan 93	03Feb 93	O.K.	9
018	Forms S Abutment	5.00	9dy	26Jan 93		16dy	26Jan 93	16Feb 93	Affected by Act 12	16
019	Falsework	0.00	0dy			25dy	01Feb 93	05Mar 93	Affected by Act 13	25
020	Deck Materials	0.00	0dy			20dy	12Feb 93	11Mar 93	Knock on Act's 12/1	20
021	Parapet Strings	0.00	0dy			45dy	23Feb 93	05May 93		45
022	Honel Drainage	7.00	4dy	06Jan 93		75dy	06Jan 93	27Apr 93	14 Week Delivery ?	71
023	NORTH ABUTMENT	14.5	14dy	19Dec 92		106dy	19Dec 92	04Jun 93		91
024	Excavation	100.	4dy	19Dec 92	04Jan 93	4dy	19Dec 92	04Jan 93		0
025	Blinding	100.	2dy	05Jan 93	06Jan 93	2dy	05Jan 93	06Jan 93		0
026	Reinforcement to Base	40.0	7dy	07Jan 93		28dy	07Jan 93	15Feb 93	Bending Schedules ?	21
027	Formwork - Conc Base	12.0	1dy	17Jan 93		24dy	17Jan 93	18Feb 93	Affected by Act' 26	23
028	Reverse Blinding	0.00	0dy			10dy	19Feb 93	04Mar 93		10
029	Reinfmt to Walls	0.00	0dy			15dy	05Mar 93	25Mar 93		15
030	Wall Pour "1"	0.00	0dy			18dy	26Mar 93	28Apr 93		18
031	Wall Pour "2"	0.00	0dy			11dy	29Apr 93	14May 93		11
032	Wall Pour "3"	0.00	0dy			14dy	17May 93	04Jun 93		14
033	SOUTH ABUTMENT	4.12	4dy	05Jan 93		123dy	05Jan 93	05Jul 93		119
034	Excavation	100.	3dy	05Jan 93	07Jan 93	3dy	05Jan 93	07Jan 93	Ahead of Programme	0
035	Blinding	100.	1dy	08Jan 93	08Jan 93	1dy	08Jan 93	08Jan 93	Ahead of Programme	0
036	Reinforcemnet Base	0.00	0dy			18dy	16Feb 93	11Mar 93		18
037	Formwork & Conc Base	0.00	0dy			15dy	01Mar 93	22Mar 93		15
038	Reverse Blinding	0.00	0dy			10dy	23Mar 93	05Apr 93		10
039	Reinfmt to Walls	0.00	0dy			15dy	06Apr 93	26May 93		15
040	Wall Pour "1"	0.00	0dy			18dy	06May 93	01Jun 93		18
065	Place in Drainage	0.00	0dy			2dy	02Jun 93	03Jun 93		2
041	Wall Pour "2"	0.00	0dy			11dy	02Jun 93	16Jun 93		11
042	Wall Pour "3"	0.00	0dy			14dy	17Jun 93	06Jul 93		14
043	FALSEWORK	4.28	6dy	05Jan 93		178dy	05Jan 93	22Sep 93		172
044	Lay Sleepers	100.	6dy	05Jan 93	12Jan 93	6dy	05Jan 93	12Jan 93	O.K.	0
045	Deck Span	0.00	0dy			18dy	08Mar 93	31Mar 93		18
046	Part Strike & Prop	0.00	0dy			2dy	13Aug 93	16Aug 93		2
047	Remove Props	0.00	0dy			1dy	01Sep 93	01Sep 93		1
048	Complete Strike	0.00	0dy			1dy	22Sep 93	22Sep 93		1
049	DECK SPAN	0.00	0dy			36dy	18Jun 93	05Aug 93		36
050	Deck Out Span	0.00	0dy			18dy	18Jun 93	13Jul 93		18
051	Rebar to Deck	0.00	0dy			22dy	06Jul 93	04Aug 93		22
052	Clean & Concrete	0.00	0dy			2dy	05Aug 93	06Aug 93		2
053	EAST CANTILEVER	0.00	0dy			16dy	09Aug 93	27Aug 93		16
054	Reinforcement	0.00	0dy			5dy	09Aug 93	13Aug 93		5
055	Forms & Concrete	0.00	0dy			12dy	12Aug 93	27Aug 93		12
056	WEST CANTILEVER	0.00	0dy			22dy	16Aug 93	15Sep 93		22
057	Reinforcement	0.00	0dy			5dy	16Aug 93	20Aug 93		6
058	Forms & Concrete	0.00	0dy			12dy	31Aug 93	15Sep 93		12
059	SLABS & FINALS	0.00	0dy			82dy	29Sep 93	28Jan 94		82
060	Backfill Abutments	0.00	0dy			6dy	29Sep 93	06Oct 93		6
061	Stabilization Period	0.00	0dy			45dy	07Oct 93	08Dec 93		45
062	Construct Slab	0.00	0dy			28dy	09Dec 93	25Jan 94		28
063	Final Snag & Clear	0.00	0dy			3dy	26Jan 94	28Jan 94		3
064	Handover B6	0.00	0dy			0dy	28Jan 94	28Jan 94		0

APPENDIX B

Deterministic Activity Network Temporal Analysis
(Activity-on-the-Arrow)

B.1 DEFINITIONS

t single estimate of mean activity duration time
T_E earliest event occurrence time
T_L latest allowable event occurrence time
ES earliest activity start time
EF earliest activity finish time
LS latest allowable activity start time
LF latest allowable activity finish time

B.2 TOTAL ACTIVITY SLACK (S)

$S = LF - EF = T_L - EF$ (T_L of the activity's successor event)

The amount of time by which the actual completion time of an activity can exceed its earliest expected completion time without directly affecting the duration of the project.

B.3 ACTIVITY FREE SLACK (S_F)

$S_F = T_E - EF$ (T_E of the activity's successor event)

The amount of time by which the actual completion time of an activity can exceed its earliest expected completion time without affecting any other activity or event in the network.

B.4 INTERFERING SLACK (IS)

$IS = S - S_F$

The amount of flexibility in scheduling an activity that must be shared with succeeding activities.

B.5 INDEPENDENT SLACK (IndS)

$IndS = T_E - T_L - t$ (T_E of the successor event, T_L of the predecessor event)

Slack that can be used exclusively for an activity having no affect on preceding or succeeding activities.

B.6 FORWARD PASS RULES

Rules	*Formulae*
1. The earliest occurrence time of an initial event in the network is taken as zero.	$T_E = 0$ (for the initial event)
2. Each activity begins as soon as its predecessor event occurs.	$ES = T_E$ (for predecessor event) $EF = ES + t = T_E + t$
3. The earliest event time is the largest of the earliest finish times of the activities merging to the event in question.	T_E = largest of $(EF_1, EF_2, \ldots, EF_n)$ for an event with n merging activities

B.7 BACKWARD PASS RULES

Rules	*Formulae*
1. The latest allowable occurrence time of a terminal event is set equal to the earliest occurrence time computed in the forward pass.	$T_L = T_E$ (for terminal event)

2. The latest allowable start time for an activity is its successor event latest allowable time minus the duration of the activity.

$LF = T_L$ (for successor event)
$LS = LF - t = T_L - t$

3. The latest allowable start time for an event is the smallest of the latest allowable start times of the activities bursting from the event in question.

T_L = smallest of $(LS_1, LS_2, \ldots, LS_n)$ for an event with n bursting activities

APPENDIX C

Pure Exclusive-Or Generalised Activity Network Analysis Using Flowgraph Theory

C.1 INTRODUCTION

This appendix deals with the analysis of pure Exclusive-Or Generalised Activity Networks. In these definitions an Activity-on-the-Arrow network is analysed.

A Generalised Activity Network consisting only of Exclusive-Or SNL input and output nodes (an XOrGAN) can be analysed using flowgraph theory. The input side of nodes in this case are Blank-Or. XOrGANs represent open flowgraphs with a single source node *s* and a single sink node *z*. A network consists of N arcs. Each arc, i, is associated with an independent, nonnegative random variable X_i (i = 1, 2, . . . , N) and a probability, p_i, representing the probability that an arc, i, is performed given that its source node is realised. Each X_i has a known distribution function $f_i(t)$ on $[0,\infty)$. Between *s* and *z* there are K paths P_j (j=1, 2, . . . , K). Arc $i \in P_j$ if arc i lies on path P_j.

A First Order Loop is defined as a path in which the start node is also the finish node. An n^{th} Order Loop is defined as *n* non-touching first order loops. In flowgraph theory the value of a *transmittance* through each path or loop is the product of the transmittances within that route.

One of the most important equations within flowgraph theory is the Topological Equation (C.1).

$$H = 1 - \sum_{i=1}^{\infty} (-1)^i L_i \qquad (C.1)$$

where $L_i = \sum i^{th}$ order loops

In a closed flowgraph (one composed entirely of loops) H = 0. In open flowgraphs, however, $H \neq 0$ and is used to calculate T, the transmittance through a network between nodes *s* and *z*, using Mason's Rule (C.2).

$$T = \frac{\left[\sum_{i=1}^{K} \left(P_i \times \sum \text{non-touching loops} \right) \right]}{H} \qquad \text{(C.2)}$$

$$\text{where } \sum \text{non-touching loops} = 1 + \sum_{j=1}^{\infty} (-1)^j l_j$$

and l_j is the sum of the j^{th} order loops not touching P_i.

C.2 MOMENT GENERATING FUNCTIONS

Mason's Rule and the Topological Equation apply to networks in which the transmittances along arcs are multiplicative. Although this is the case for arc probabilities in Generalised Activity Networks, the time and cost elements are additive. In order to enable the time element to be handled by flowgraph theory it requires transforming into a multiplicative form. Pritsker and Happ (1966) proposed using Moment Generating Functions (MGFs) for this operation. Their reasoning behind this was that the resultant transmittances 'form a system of linear independent equations and, hence, can be reduced to a single W-function topology equation of flowgraph theory'.

If e represents the exponential constant, then the time associated with arc i is characterised by the MGF:

$$M_i(s) = \int_0^{\infty} e^{st} f_i(t)\, dt \qquad \text{(C.3)}$$

The distribution function, $f_i(t)$, can take many forms all of which can be converted into the corresponding MGF using equation C.3. For example, equation C.4 represents the MGF of a constant duration t_o. The MGF of the Normal distribution with mean μ and standard deviation σ is given in equation C.5, and equation C.6 represents the MGF of the Uniform distribution with limits a and b.

$$M_i(s) = e^{st_o} \qquad (C.4)$$

$$M_i(s) = e^{s\mu + \frac{1}{2}s^2\sigma^2} \qquad (C.5)$$

$$M_i(s) = \frac{e^{sb} - e^{sa}}{(b - a)s} \qquad (C.6)$$

The second stage of the transformation into multiplicative form is to generate the ω-function, $\omega_i(s)$, of arc i (C.7).

$$\omega_i(s) = p_i M_i(s) \qquad (C.7)$$

By converting each arc duration into an equivalent ω-function one can employ Mason's Rule to reduce an XOrGAN into a single equivalent arc, e, between nodes *s* and *z* (figure C.1).

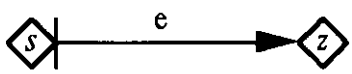


Figure C.1 Equivalent Arc

If one analyses this equivalent arc then one has the time characteristics of an entire network. $\omega_e(s)$ represents the ω-function of this equivalent arc. The arc has probability, p_e, associated with it and MGF of the time distribution, $M_e(s)$. Thus:

$$\omega_e(s) = p_e M_e(s) \qquad (C.8)$$

As all MGFs reduce to unity at s=0 the value of p_e can be calculated from:

$$\omega_e(0) = p_e M_e(0) = p_e \qquad (C.9)$$

Rearranging equations C.8 and C.9 leads to:

$$M_e(s) = \frac{\omega_e(s)}{\omega_e(0)} \qquad (C.10)$$

The n^{th} derivative of $M_e(s)$ evaluated at s=0 yields the expected value of the n^{th} power of t_e - the duration of the equivalent arc. The second derivative enables the standard

deviation of this duration to be calculated. Equations C.11 and C.12 show these derivatives.

$$E(t_e) = \frac{d}{ds} M_e(s)\Big|_{s=0} \qquad (C.11)$$

$$E(t_e^n) = \frac{d^n}{ds^n} M_e(s)\Big|_{s=0} \qquad (C.12)$$

Thus, the required time characteristics of the network can be calculated.

C.3 EXAMPLE

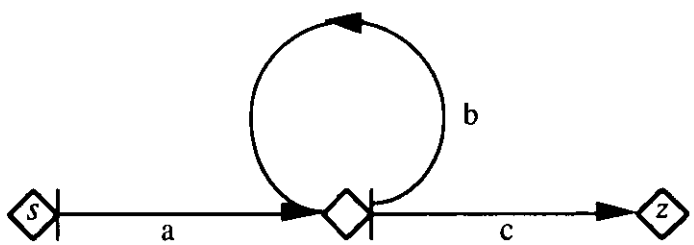


Figure C.2 Example Network

As an example take the simple XOrGAN shown in figure C.2. Each arc has associated with it the following characteristics:

Arc	Probability	Time Distribution Function
a	$p_a = 1$	Constant, $f_a(t) = t_a$
b	$p_b = 1 - p_c$	Negative Exp, $f_b(t) = \lambda e^{-\lambda t}$
c	$p_c = 1 - p_b$	Constant, $f_c(t) = t_c$

The ω-function of the time transmittances through each arc a, b, c are ω_a, ω_b and ω_c respectively and, from equations (C.3) and (C.7), are evaluated as:

$$\omega_a = e^{st_a} \qquad \omega_b = \frac{p_b \lambda}{\lambda - s} \qquad \omega_c = p_c e^{st_c}$$

From Mason's rule the equivalent transmittance, T, between nodes *s* and *z* is:

$$T = \frac{\omega_a \omega_c}{1 - \omega_b} = \omega_e(s)$$

$$\text{Now } p_e = \omega_e(0) = \frac{p_c}{1 - p_b} = 1$$

$\therefore$ $M_e(s) = \omega_e(s)$ and thus

$$E(t_e) = \frac{d}{ds} M_e(s)\Big|_{s=0} = t_a + t_c + \frac{p_b}{(1-p_b)\lambda}$$

and

$$\sigma_e^2 = E(t_e^2) - E(t_e)^2 = \frac{1}{(1 - p_b)^2 \lambda^2} - \frac{1}{\lambda^2}$$

from (C.11) and (C.12).

APPENDIX D

The Minimum of a Finite Set of Normal Random Variables

D.1 INTRODUCTION

This appendix is based on the work presented in Dawson (1993 and 1994c) which are developments from Clark's work [Clark 1961b].

X, Y, and Z are three independent Normal distributions with means μ_x, μ_y, μ_z and standard deviations σ_x, σ_y, σ_z respectively. Equations D.1 and D.2 are defined from Clark (1961b). ρ denotes the coefficient of linear correlation between X and Y ($\rho(X,Y)$).

$$a^2 = \sigma_x^2 + \sigma_y^2 - 2\sigma_x\sigma_y\rho \qquad (D.1)$$

$$\alpha = \frac{\mu_x - \mu_y}{a} \qquad (D.2)$$

If $\rho(\,)$ denotes the coefficient of linear correlation then

$$\rho_1 = \rho(X,Z) \qquad (D.3)$$

$$\rho_2 = \rho(Y,Z) \qquad (D.4)$$

Equations D.5 and D.6 (adapted from Clark (1961b)) represent the first two moments (v_1 and v_2) about the origin of the distribution of the maximum of the two distributions X and Y. $\Phi(x)$ represents the standard Normal integral and $\Psi(x)$ the standard Normal distribution function.

$$v_1 = \Phi(\alpha)\,(\mu_x - \mu_y) + \mu_y + a\Psi(\alpha) \qquad (D.5)$$

$$v_2 = \Phi(\alpha)\,(\mu_x^2 - \mu_y^2 + \sigma_x^2 - \sigma_y^2) + \mu_y^2 + \sigma_y^2 + (\mu_x + \mu_y)a\Psi(\alpha) \qquad (D.6)$$

It follows that the standard deviation (σ_{max}) of this maximum distribution can be defined as shown in equation D.7.

$$\sigma_{max} = \sqrt{v_2 - v_1^2} \qquad \text{(D.7)}$$

Clark went on to show how the maximum of n distributions could be calculated by employing equations D.8 and D.9.

$$\max(X,Y,Z,\ldots) = \max(\max(\max\ldots(((X,Y),Z),\ldots)) \qquad \text{(D.8)}$$

$$\rho(Z,\ \max(X,Y)) = \frac{\Phi(\alpha)\ (\sigma_x\rho_1 - \sigma_y\rho_2) + \sigma_y\rho_2}{\sqrt{v_2 - v_1^2}} \qquad \text{(D.9)}$$

D.2 MINIMUM

Although Clark leaves his work at the maximum of a finite set of random variables it does lead to some equally important results. Adapting equations D.5 and D.6 produces formulae for calculating the first and second moments of the distribution of the *minimum* of a finite set of Normal random variables (equations D.10 and D.11).

$$v_1 = \Phi(\alpha)\ (\mu_y - \mu_x) + \mu_x - a\Psi(\alpha) \qquad \text{(D.10)}$$

$$v_2 = \Phi(\alpha)\ (\mu_y^2 - \mu_x^2 + \sigma_y^2 - \sigma_x^2) + \mu_x^2 + \sigma_x^2 - (\mu_x + \mu_y)a\Psi(\alpha) \qquad \text{(D.11)}$$

Similarly, equations D.12 and D.13 represent the equivalent formulae required to calculate the minimum of three or more Normal random variables.

$$\min(X,Y,Z,\ldots) = \min(\min(\min\ldots(((X,Y),Z),\ldots)) \qquad \text{(D.12)}$$

$$\rho(Z,\ \min(X,Y)) = \frac{\Phi(\alpha)\ (\sigma_y\rho_2 - \sigma_x\rho_1) + \sigma_x\rho_1}{\sqrt{v_2 - v_1^2}} \qquad \text{(D.13)}$$

Equation D.14, from Clark (1961b), is provided for completeness. Both distributions A and B are independent of the other three distributions in this equation. The proof is

provided in Clark (1961b).

$$\rho(X+A, Y+B) = \frac{\rho(X,Y)\sqrt{V(X)}\sqrt{V(Y)}}{\sqrt{V(X+A)}\sqrt{V(Y+B)}} \qquad (D.14)$$

D.3 EXAMPLE

D.3.1 Overview

Take, as an example, the problem of finding the shortest route through a directed, acyclic network in which the distribution times of the arcs are Normally distributed with known means and standard deviations. For the minimum duration we assume that events occur as soon as their earliest incoming arc completes, ie node inputs are represented by C-Or in this case. Figure D.1, adapted from Clark (1961b), represents just such a network. It is intended to find the distribution parameters of the minimum event time of node D. Thus, one is interested in calculating the mean (E(D)) and variance (V(D)) of this minimum completion time.

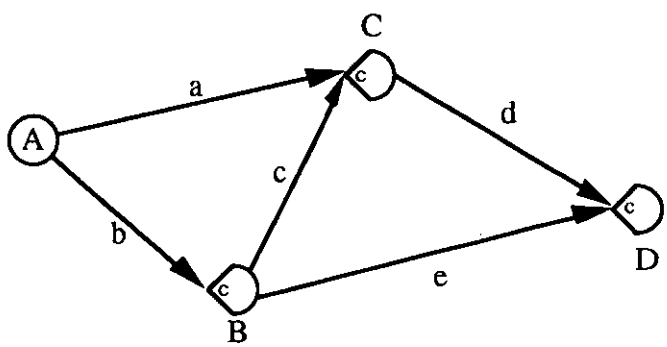


Figure D.1 Example Network

D.3.2 Solution

Clearly E(A) = 0, V(A) = 0

E(B) = E(A)+E(b) = E(b) known
V(B) = V(A)+V(b) = V(b) known

The event time of C is distributed as the minimum of (a, B+c). To calculate E(C) and V(C), E(a) and V(a) are required which are known.

E(B+c) = E(B) + E(c) and V(B+c) = V(B) + V(c) are also required which are again all known. ρ(a, B+c) = 0 as 'a' is independent of B+c. Therefore, by employing equations D.1, D.2, D.10, and D.11, both E(C) and V(C) can be calculated.

It only remains to calculate the expected completion time of event D (E(D)) and the variance (V(D)) of this time. The event time of D is distributed as the minimum of (C+d, B+e). The estimates of the expected values and variances of C+d and B+e are similar to those made earlier. By employing equations D.13 and D.14 in a manner similar to Clark, it is also possible to calculate ρ(C+d, B+e). Thus E(D) and V(D) can be calculated directly.

APPENDIX E

Typical Output from the BestFit™ Data Analysis Package

Statistics for PoisBL1 [High Precedence, Low Parallelism, One Hundred Activities, Poisson Activity Temporal Functions]

Minimum = 74.0
Maximum = 128.0
Mode = 97.0
Mean = 99.314
Std Deviation = 9.557654
Variance = 91.348753
Skewness = 0.066171
Kurtosis = 3.067222
Input Settings:
Type of Fit: Full Optimisation
Tests Run: Chi-square K-S Test A-D Test
Histogram:
Min: 74.0
Max: 128.0
P1.0: 2.006018e-3
P2.0: 2.006018e-3
P3.0: 2.006018e-3
P4.0: 2.006018e-3
P5.0: 4.012036e-3
P6.0: 1.003009e-3
P7.0: 6.018054e-3
P8.0: 5.015045e-3
P9.0: 0.01003
P10.0: 6.018054e-3
P11.0: 0.017051
P12.0: 0.015045
P13.0: 7.021063e-3
P14.0: 0.02006
P15.0: 0.016048
P16.0: 0.022066
P17.0: 0.038114

P18.0: 0.037111
P19.0: 0.027081
P20.0: 0.032096
P21.0: 0.031093
P22.0: 0.036108
P23.0: 0.042126
P24.0: 0.054162
P25.0: 0.048144
P26.0: 0.036108
P27.0: 0.036108
P28.0: 0.042126
P29.0: 0.035105
P30.0: 0.039117
P31.0: 0.036108
P32.0: 0.031093
P33.0: 0.033099
P34.0: 0.033099
P35.0: 0.026078
P36.0: 0.027081
P37.0: 0.019057
P38.0: 0.022066
P39.0: 0.017051
P40.0: 8.024072e-3
P41.0: 0.019057
P42.0: 0.01003
P43.0: 0.012036
P44.0: 5.015045e-3
P45.0: 3.009027e-3
P46.0: 7.021063e-3
P47.0: 3.009027e-3
P48.0: 3.009027e-3
P49.0: 2.006018e-3
P50.0: 2.006018e-3
P51.0: 2.006018e-3
P52.0: 1.003009e-3
P53.0: 1.003009e-3
P54.0: 3.009027e-3
P55.0: 2.006018e-3

#Classes = 55.0

Filtering:
X minimum 74.0
X maximum 128.0

BestFit Results

Function	C-S Test	Rank	K-S Test	Rank	A-D Test	Rank
Poisson(99.31)	0.052878	1	N/A		N/A	
Erlang(107,0.93)	0.052969	2	0.036854	3	0.792257	3
Gamma(108,0.92)	0.053475	3	0.0362	2	0.776855	2
Lognormal(99.32,9.66)	0.054618	4	0.042639	5	1.135987	4
Logistic(99.03,5.84)	0.055024	5	0.035524	1	1.765252	5
Normal(99.31,9.56)	0.059323	6	0.037459	4	0.717108	1
Weibull(9.67,103)	0.144159	7	0.095192	7	13.849956	7
Chisq(99)	0.193964	8	0.132912	8	34.21684	8
NegBin(1,0.00997)	6.550607	9	N/A		N/A	
Geomet(0.00997)	6.550607	10	N/A		N/A	
Beta(0.21,0.87)*54+74	449268.2	11	0.80116	9	1691.95918	9
Binomial(128,0.78)	5.445e+8	12	N/A		N/A	
Triang(74,97.6,128)	1e+34	13	0.078903	6	10.790222	6
HyperGeo(135,129,292)	1e+34	14	N/A		N/A	

APPENDIX F

Generating Pseudo Random Numbers from Known Distributions

F.1 INTRODUCTION

Almost all computers provide a means of generating pseudo random numbers from a 'continuous' Rectangular distribution of some finite range. The statistician, however, often needs to simulate samples from other, more popular distributions. The algorithms presented represent ways of generating random samples from known continuous and discrete distributions given that a random sample is available from a continuous Rectangular distribution on [0,1].

U_i (i = 1, 2, . . . , N) are N independent random samples from the Rectangular distribution on [0,1].

X represents one sample from the distribution in question.

F.2 CONTINUOUS DISTRIBUTIONS

F.2.1 Normal Distribution [Rubinstein 1981, p90]

$N(\mu,\sigma^2)$ is Normally distributed with mean, μ, and variance, σ^2.

$$Y = \sum_{i=1}^{12} U_i - 6$$

$$X = \mu + \sigma Y$$

F.2.2 Lognormal Distribution [Law and Kelton 1991, p492]

$LN(\mu,\sigma^2)$ is a Lognormal distribution with mean μ and variance σ^2.

$$\mu_1 = \ln\left(\frac{\mu^2}{\sqrt{\sigma^2 + \mu^2}}\right) \qquad \sigma_1^2 = \ln\left(\frac{(\sigma^2 + \mu^2)}{\mu^2}\right)$$

Generate $Y \sim N(\mu_1, \sigma_1^2)$

$X = e^Y$

F.2.3 Rectangular Distribution

R(a,b) is a Rectangular distribution with range [a,b].

$X = U_1(b - a) + a$

F.2.4 Triangular Distribution (adapted from Law and Kelton (1991), p494)

T(a,m,b) is a Triangular distribution with range [a,b] and mode m.

$Y = U_1(b - a)$

If $Y \leq (m - a)$ then $X = a + \sqrt{Y(m - a)}$

otherwise $X = b - \sqrt{(b - m)(b - a - Y)}$

F.2.5 Gamma Distribution [Cheng 1977]

$G(\alpha, \beta)$ is Gamma distributed with shape parameter, α (α>1), and scale parameter, β.

Initialise $a = \frac{1}{\sqrt{2\alpha - 1}}$, $b = \alpha - \ln 4$, $q = \alpha + \frac{1}{a}$, $\theta = 4.5$, $d = 1 + \ln \theta$

1. Generate U_1 and U_2

$$V = a \ln\left(\frac{U_1}{1 - U_1}\right), \qquad Y = \alpha e^V, \qquad Z = U_1^2 U_2, \qquad W = b + qV - Y$$

2. If $W + d - \theta Z \geq 0$ **or** $W \geq \ln Z$ then $X = Y$
otherwise return to step 1.

F.2.6 Beta Distribution [Rubinstein 1981, p81]

$\beta(\alpha_1, \alpha_2)$ is Beta distributed with parameters α_1 and α_2.

Generate $Y_1 \sim G(\alpha_1, 1)$
$Y_2 \sim G(\alpha_2, 1)$

$$X = \frac{Y_1}{(Y_1 + Y_2)}$$

F.3 DISCRETE DISTRIBUTIONS

F.3.1 Bernoulli Distribution [Law and Kelton 1991, pp496-497]

Bern(p) is a Bernoulli distribution that represents a random occurrence of two possible outcomes. p is the probability of one outcome.

if $U_1 \leq p$ then X=1
otherwise X=0

F.3.2 Discrete Uniform Distribution [Law and Kelton 1991, p497]

DU(i,j) is a Discrete Uniform distribution on [i,j].

$X = i + \lfloor U_1(j - i + 1) \rfloor$

F.3.3 Poisson Distribution (adapted from Ahrens and Deiter (1974))

$P(\lambda)$ is a Poisson distribution with mean λ.

Initialise $L = e^{-\lambda}$, $p = 1$, $k = -1$

repeat

$p = p\ U_{k+2}$

$k = k + 1$

until $p < L$

$X = k$

F.3.4 Binomial Distribution [Law and Kelton 1991, p502]

Bin(t,p) is a Binomial distribution representing the number of successes of probability p in t independent Bernoulli trials.

Generate $Y_i \sim Bern(p)$ $\quad (i = 1, 2, 3, \ldots, t)$

$$X = \sum_{i=1}^{t} Y_i$$

F.3.5 Geometric Distribution [Law and Kelton 1991, p502]

Geom(p) is a Geometric distribution representing the number of failures before a success of probability p occurring in a sequence of independent Bernoulli trials.

$$X = \left\lfloor \frac{\ln U_1}{\ln(1 - p)} \right\rfloor$$

F.3.6 Negative Binomial Distribution [Law and Kelton 1991, p503]

NB(s,p) is a Negative Binomial distribution representing the number of failures before the s^{th} success of probability p in a sequence of independent Bernoulli trials.

Generate $Y_i \sim \text{Geom}(p)$ $\quad (i = 1, 2, 3, \ldots, s)$

$$X = \sum_{i=1}^{s} Y_i$$

Appendix G

Selected Company Addresses

Aran Ltd
Rivermead, Pipers Way,
Thatcham, Berks
RG13 4EP

Asta Development Corporation Ltd
5 St. Andrew's Court
Wellington Street
Thames, Oxon
OX9 3WT

British Telecom
Martlesham Heath
Ipswich
Suffolk

Complete Project Management Ltd
Dovetail House
Wycombe Road
Stokenchurch
Bucks
HP14 3RQ

Computer Associates Plc
183/187 Bath Road
Slough, Berks
SL1 4AA

Deepak Sareen Associates
Bydell House
Sudbury Hill
Harrow–on–the–Hill
Middlesex
HA1 3NJ

GPT Telecommunications Limited
Beeston
Nottingham

Hoskyns Group Plc
Hoskyns South Bank
95 Wandsworth Road
London
SW8 2HG

IBM (UK) Ltd
76 Upper Ground
London
SE1 9PZ

John Cockerham and Associates
Slington House
Rankine Road
Daneshill Estate West
Basingstoke
Hampshire
RG24 0PH

LBMS
Evelyn House
62 Oxford Street
London
WIN 9LF

Lucas Management Systems
Artemis House
219 Bath Road
Slough
Berkshire
SL1 4AA

Leach Management Systems
Temple House
6–7 The Causeway
Chippenham
Wilts
SN15 3BT

Mantix Systems Ltd
Mantix House
London Road
Bracknell
Berks
RG12 2XH

Microplanning International Ltd
34 High Street
Westbury–on–Trym
Bristol
BS9 3DZ

Microsoft
Microsoft Place
Winnersh Triangle
Wokingham
Berks
RG11 5TP

OPL
5 The Spinney
Tattenham Corner
Epsom
Surrey
KT18 5QX

Palisade Corporation
31 Decker Road
Newfield
New York
14867

PMP (Project Management Professional Services Ltd)
PMP House
Gardner Road
Maidenhead
Berks
SL6 7RJ

PSDI (UK) Ltd
No 5 Woking Eight
Forsyth Road
Woking
Surrey
GU21 5SB

Panorama Software Corporation Ltd
P O Box 2128
Walton Court
Station Avenue
Walton on Thames
Surrey
KT12 1YG

Pariss Ltd
Premier House
15 Wheeler Gate
Nottingham
NG1 2NN

People in Technology Ltd
Epworth House
25 City Road
London
EC1Y 1AA

Primavera Systems Inc
Unit 2
2nd Floor
Elsinore House
77 Fulham Palace Road
Hammersmith
London
W6 8JA

Risk Decisions Ltd
27 Park End Street
Oxford
OX1 1HU

Scitor Corporation
393 Vintage Park Drive
Suite 140
Foster City
CA 94404

TBV Consult
The Lansdowne Building
Lansdowne Road
Croydon
CRO 2BX

Welcom Software Technology International
South Bank Technopark
90 London Road
London
SE1 6LN

APPENDIX H

Questionnaire Sent to a Number of Midlands' Companies

1 What kinds of developments are you involved with?
2 How many person days (or months) are involved in a typical project?
3 Typically how many staff are involved?
4 Typically how many separate tasks do you break a project into for planning purposes?
5 Do you use any project management tools for planning (Y/N)?
6 If so which tool(s) do you use (for example PMW, InstaPlan, Open Plan, Cascade)?
7 If not -
    7.1 Why not?
    7.2 How do you decide when to do things?
    7.3 How do you estimate project costs and durations?
8 Who is involved with planning your projects?
9 Do you attempt to assess risks (cost and time uncertainties) in your projects (Y/N)?
10 If so - what methods do you use?
11 Do you find your current methods allow you to adequately ascertain the risks of any cost and time overruns in a project?
12 Do you use simulation to analyse your project plans (Y/N)?
13 If so - how do you decide how many simulations to perform?

Below are some theoretical ideas for project management tools that offer certain facilities:

a) An ability to plan for repetitive tasks by allowing managers to represent these in project plan networks (PERT networks) as loops.
b) An ability to identify project risk points more visibly in PERT networks.
c) An ability for PERT networks to modify themselves dynamically as information is provided form earlier phases in the project life cycle.
d) An ability to plan, and hence evaluate, alternative project outcomes (including terminating the project).
e) An ability to model activities using several different kinds of cost and time functions.
f) An ability to plan activities that depend on previous outcomes in the project.

References

Abdel-Hamid, T.K. (1988) 'Understanding the "90% Syndrome" in Software Project Management: A Simulation-Based Case Study', The Journal of Systems and Software, Vol 8, pp319-330

Adlakha, V.G. (1987) 'A Monte Carlo Technique with Quasirandom Points for the Stochastic Shortest Path Problem', American Journal of Mathematical and Management Sciences, Vol 7(3 and 4), pp325-358

Adlakha, V.G. (1992) 'An Empirical Evaluation of Antithetic Variates and Quasirandom Points for Simulating Stochastic Networks', Simulation, Vol 58(1), pp23-31

Adlakha, V.G. and Arsham, H. (1992) 'A Simulation Technique for Estimation in Perturbed Stochastic Activity Networks', Simulation, Vol 58(4), pp258-267

Adlakha, V.G. and Kulkarni, V.G. (1989) 'A Classified Bibliography of Research on Stochastic PERT Networks: 1966-1987', INFOR (Canada), Vol 27(3), pp272-296

Adler, R. (1992) 'Blackboard Systems', Encyclopedia of Artificial Intelligence, Ed. Shapiro, S. C., Vol 1, John Wiley and Sons Incorporated, New York, pp116-126

Agresti, W.W. (1986a) 'New Paradigms for Software Development', IEEE Computer Society Tutorial EH0245–1, ISBN 0–8186–0707–6

Agresti, W.W. (1986b) 'Framework for a Flexible Development Process', IEEE Computer Society Tutorial, EH0245-1, ISBN 0-8186-0707-6

Ahmad, M. Opdyke, W.F. Kim, M.W. and Zislis, M.W. (1988) 'A Knowledge Based Approach to Assist in Telecommunications Software Project Management', IEEE International Conference on Communications, pp1455-1458

Ahrens, J.H. and Dieter, U. (1974) 'Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions', Computing, Vol 12(3), pp223-246

Alavi, M. (1984) 'An Assessment of the Prototyping Approach to Information Systems Development', Communications of the ACM, Vol 27(6), pp556-563

Allen, J.F. (1983) 'Maintaining Knowledge About Temporal Intervals', Communications of the ACM, Vol 26(11), pp832-843

Allen, J.F. and Hayes, P.J. (1985) 'A Common-Sense Theory of Time', 9th International Conference on AI, IJCAI-85, pp528-531

Anderson, T.W. (1971) 'The Statistical Analysis of Time Series', John Wiley and Sons Incorporated, New York

Anklesaria, K.P. and Drezner, Z.V.I. (1986) 'A Multivariate Approach to Estimating the Completion Time for PERT Networks', Journal of the Operational Research Society, Vol 37(8), pp811-815

Badiru, A.B. (1991) 'A Simulation Approach to PERT Network Analysis', Simulation, Vol 57(4), pp245-255

Ballot, M. (1989) 'Probability and PERT Avoiding Errors Through Simulation', Modelling and Simulation on Microcomputers, Society for Computer Simulation, International Proceedings of the SCS Western Multi Conference, San Diego, CA, USA, 4-6 Jan, pp14-18

Barnes, D. and Brown, P. Ed. (1986) 'Software Engineering 86', IEE Computing Series 6, Peter Peregrinus

Barnes, M. (1989) in 'Have Project Will Manage', BBC2

Beierle, C. Olthoff, W. and Voss, A. (1986) 'Towards a Formalisation of the Software Development Process', in Barnes and Brown (1986)

Bellas, C.J. and Samli, A.C. (1973) 'Improving New Product Planning with GERT Simulation', California Management Review, Vol 15(4), pp14-21

Benington, H.D. (1956) 'Production of Large Computer Programmes', Proceedings ONR, Symposium on Advanced Programming Methods, pp15-27

Berman, E.B. (1964) 'Resource Allocation in a PERT network Under Continuous Activity Time-Cost Functions', Management Science, Vol 10(4), pp734-745

Blaney, J. (1989) 'Managing Software Development Projects', Proceedings of the Project Management Institute Annual Seminar, Atlanta, USA, 7-11 October, pp410-417

Boehm, B.W. (1981) 'Software Engineering Economics', Prentice-Hall Incorporated, Englewood Cliffs, New Jersey

Boehm, B.W. (1984) 'Software Engineering Economics', IEEE Transactions on Software Engineering, Vol 10(1), pp4-21

Boehm, B.W. (1988) 'A Spiral Model for Software Development and Enhancement', Computer, Vol 21(5), pp61 -72

Bowen, R.B. (1990) 'Software Project Management Under Incomplete and Ambigious Specifications', IEEE Transactions Engineering Management (USA), Vol 37(1), pp10-21

Brachman, R.J. and Levesque, H.J. Ed. (1985) 'Readings in Knowledge Representation', Magan Kaufmann Publishers, Los Altos

Brooks, F.P. (1987) 'No Silver Bullet, Essence of Accidents of Software Engineering', Computer, Vol 20(4), pp10-19

Brown, A. W. (1988) 'Integrated Project Support Environments', Information and Management, Vol 15(3), pp125-134

Brown, A.W. Robinson, D.S. and Weedon, R. (1986) 'Managing Software Development', in Barnes and Brown (1986), pp197-225

Brown, R. (1988) 'Knowledge-Based Scheduling and Resource Allocation in the CAMPS Architecture', Intelligent Manufacturing, The Benjamin/Cummings Publishing Company Incorporated, California, pp165-186

Burgess, A.R. and Killebrew, J.B. (1962) 'Variation in Activity Level on a Cyclical Arrow Diagram', Journal of Industrial Engineering, Vol 13(2), pp76-83

Burt, J.M. and Garman, M.B. (1971a) 'Conditional Monte Carlo: A Simulation Technique for Stochastic Network Analysis', Management Science, Vol 18(3), pp207-217

Burt, J.M. and Garman, M.B. (1971b) 'Monte Carlo Techniques for Stochastic PERT Network Analysis', INFOR, Vol 9(3), pp248-262

Burt, J.M. Gaver, D.P. and Perlas, M. (1970) 'Simple Stochastic Networks: Some Problems and Procedures', Naval Research Logistics Quarterly, Vol 17, pp439-458

Butler Group (1993) 'Dynamic Systems Development', Concept Papers Number 5, September, Hull

Carter, G.D. Clare, C.P. and Thorogood, D.C.J. (1987) 'Engineering Project Management Techniques and their Application to Computer Projects', Software Engineering Journal, January, pp15-20

Chae, K.C. and Kim, S. (1990) 'Estimating the Mean and Variance of PERT Activity Time Using Likelihood-ratio of the Mode and the Midpoint', IIE Transactions, Vol 22(3), pp198-202

Chapman, K.P. and Manesero, A. (1988) 'An Intelligent Knowledge-Based System for Construction Project Management', European Coordinating Committee for AI Expert Systems and Their Applications, 8th International Workshop, pp505-514

Cheng, R.C.H. (1977) 'The generation of Gamma variables with non-integral shape parameters', Applied Statistics, Vol 26, pp71-75

Clark, C.E. (1961a) 'The Optimum Allocation of Resources Among Activities of a Network', Journal of Industrial Engineering, Vol 12, pp11-17

Clark, C.E. (1961b) 'The Greatest of a Finite Set of Random Variables', Operations Research, Vol 9(2), pp145-162

Clark, K.L. and McCabe, F.G. (1982) 'Prolog: A Language for Implementing Expert Systems', Machine Intelligence, Vol 2, pp455-470

Conover, W.J. (1971) 'Practical Non-Parametric Statistics', John Wiley and Sons Incorporated, New York

Conte, S.D. Dunsmore, H.E. and Shen, V.Y. (1986) 'Software Engineering Metrics and Models', The Benjamin/Cummings Publishing Company Incorporated, California

Cook, T.M. and Jennings, R.H. (1979) 'Estimating a Project's Completion Time Distribution Using Intelligent Simulation Methods', Journal of the Operational Research Society, Vol 30, pp1103-1108

Dawson, C.W. and Dawson, R.J. (1993b) 'Simulation of Stochastic and Generalised Actvity Networks', International AMSE Conference, London, Proceedings Vol 3, pp41-55

Dawson, C.W. and Dawson, R.J. (1994a) 'A Clarification of Node Representation in Generalised Activity Networks for Practical Project Management', International Journal of Project Management, Vol 12(2), pp81-88

Dawson, C.W. and Dawson, R.J. (1994b) 'A Comparative Simulation of Pure Probabilistic and Generalised Activity Networks', Simulation, The Journal of the Society for Computer Simulation (submitted)

Dawson, C.W. and Dawson, R.J. (1994c) 'Towards More Flexible Management of Software Systems Development Using Metamodels', Software Engineering Journal (submitted)

Dean, T.L. and McDermott, D.V. (1987) 'Temporal Data Base Management', Artificial Intelligence, Vol 32(1), pp1-55

DeKleer, J. (1984) 'Choices Without Backtracking', American Association for AI, Proceedings of the National Conference on AI, August 6-10, pp79-85

DeKleer, J. (1986a) 'An Assumption-Based TMS', AI Journal, Vol 28, pp127-162

DeKleer, J. (1986b) 'Extending the ATMS', AI Journal, Vol 28, pp163-196

DeKleer, J. and Williams, B.C. (1987) 'Back to Backtracking, Controlling the ATMS', Proceedings of the 5th National Conference on AI, Magan Kaufmann Publishers, Los Altos, pp910-917

DeMarco, T. (1982) 'Controlling Software Projects: Management, Measurement and Estimation', Yourdon Monograph, Prentice-Hall Incorporated, Englewood Cliffs, New Jersey

DeMarco, T. and Lister, T. Eds. (1990) 'Software State-of-the-Art', Dorset House Publishing, New York

Department of the Navy (1958) 'Summary Report, phase 1', Special Projects Office, Washington DC

DOD (1975) 'Work Break-down Structures for Defence Material Items', Military Standard 881a, Department of Defenses, Washington DC

Dodin, B. (1986) 'Minimum Number of Arcs in Conditional Monte Carlo Sampling of Stochastic Networks', Canadian Journal of Operations Research and Information Processing, INFOR, Vol 24(1), pp33-44

Donaldson, W.A. (1964) 'The Estimation of the Mean and Variance of a PERT Activity Time', Operations Research, pp382-385

Dooley, A.R. (1964) 'Interpretations of PERT', Harvard Busines Review, March/April, pp160-168

Doyle, J. (1978) 'Truth Maintenance System for Problem Solving', International Joint Conference on Artificial Intelligence, IJCAI-78, p247

Doyle, J. (1979a) 'A Glimpse of Truth Maintenance', AI: An MIT perspective, Vol 1, MIT, pp117-135

Doyle, J. (1979b) 'A Truth Maintenance System', Artificial Intelligence, Vol 12(3), pp231-272

Dressler, O. (1988) 'Extending the Basic ATMS', Proceedings of 8th European Conference on AI, Munich, August 1-15, ISBN 0273087983, International Library, pp535-540

Drezner, S.M. and Pritsker, A.A.B. (1966) 'Network Analysis of a Count-Down', The Rand Corporation, RM-4976-NASA, Santa Monica, California

Eisner, H. (1962) 'A Generalised Network Approach to the Planning and Scheduling of a Research Project', Operations Research, Vol 10, pp115-125

Elmaghraby, S.E. (1964) 'An Algebra for the Analysis of Generalised Activity Networks', Management Science, Vol 10(3), pp494-514

Elmaghraby, S.E. (1966) 'On Generalised Activity Networks', Journal of Industrial Engineering, Vol 17(11), pp621-631

Elmaghraby, S.E. (1967) 'On the Expected Duration of PERT Type Networks', Management Science, Vol 13(5), pp299-306

Falla, M. (1991) 'A Measured Approach to Method', Computer Weekly, April, p20

Fisher, D.L. Saisi, D. and Goldstein, W.M. (1985) 'Stochastic PERT Networks: OP Diagrams, Critical Paths and the Project Completion Time', Computers and Operations Research, Vol 12(5), pp471-482

Fishman, G.S. (1985) 'Estimating Network Characteristics in Stochastic Activity Networks', Management Science, Vol 5, pp579-593

Foster, A.T. (1987) 'Artificial Intelligence in Project Management', Chartered Mechanical Engineering, Vol 34(3), pp44-46

Fulkerson, D.R. (1962) 'Expected Critcal Path Lengths in PERT networks', Operations Research, Vol 10, pp808-817

Garman, M.B. (1972) 'More on Conditional Sampling in the Simulation of Stochastic Networks', Management Science, Vol 19(1), pp90-95

Gilb, T. (1988) 'Principles of Software Engineering Management', Addison-Wesley Publishing Company, Wokingham

Golenko-Ginzburg, D. (1988) 'On the Distribution of Activity Time in PERT', Journal of the Operational Research Society, Vol 39(8), pp767-771

Goodman, L.J. and Love, R.N. Ed. (1980) 'Project Planning and Management – An Integrated Approach', Pergammon Press, New York

Gotthardt, H. and Winkelmann, R. (1983) 'Software Development', John Wiley and Sons Incorporated, New York

Grady, R.B. and Caswell, D.L. (1986) 'Software Metrics: Establishing a Company–Wide Program', Prentice-Hall Incorporated, Englewood Cliffs, New Jersey

Grey, S. (1994) Risk Manager, ICL Enterprises, Wokingham, Berks, Pers Comm

Guerrieri, E. (1987) 'Expert Systems and Prolog in Multiple Project Management', Knowledge Based Expert Systems in Engineering: Planning and Design, Southampton Computational Mecahnics Publications, pp185-209

Hagstrom, J.N. (1990) 'Computing the Probability Distribution of Project Duration in a PERT Network', Networks, Vol 20, pp231-244

Hart, A. (1989) 'Knowledge Acquisition for Expert Systems', Kogan Page

Hayes-Roth, B. (1983) 'The Blackboard Architecture: A General Framework for Problem Solving?', Report Number HPP-83-30, Department of Computer Science, Stanford University

Hayes-Roth, B. (1984) 'BB1: An Architecture for Blackboard Systems that Control, Explain and Learn About Their Own Behaviour', Report Number Stan-CS-84-1034, Department of Computer Science, Stanford University

Hayes-Roth, B. (1985a) 'A Blackboard Architecture for Control', Journal of Artificial Intelligence, Vol 26, pp251-321

Hayes–Roth, B. (1985b) 'A BB Architecture for Control', Artificial Intelligence – An International Journal, Vol 26(3)

Hays, W.L. (1988) 'Statistics', 4th Edition, Holt, Rinehart and Winston Incorporated, New York

Healey, T.L. (1960) 'Activity Subdivision and PERT Probability Statements', Operations Research, pp341-348

Herbert, P.J. Hinde, C.J. Bray, A.D. Launders, V.A. Round, D. and Temple, D.M. (1990) 'Feature Recognition within a Truth Maintained Process Planning System', International Journal Computer Integrated Manufacturing, Vol 3(2), pp121-132

Hill, A. (1990) 'A Bad Case of Malaise in UK Management', Computer Weekly, 18 October, p20

Howes, N.R. (1984) 'Managing Software Development Projects for Maximum Productivity', IEEE Transactions on Software Engineering, Vol SE10(1), pp27-35

Hurley, D. (1993) 'PM at the Heart of the BT Network', Project Manager Today, June, pp12-16.

Iman, R.L. Davenport, J.M. and Zeigler, D.K. (1980) 'Latin Hypercube Sampling (A Programmers Guide)', Technical Report SAND79-1473, Sandia Laboratories, Albuquerque

Interrante, L.D. and Biegal, J.E. (1991) 'A Modified GERT Network for Automatic Acquisition of Temporal Knowledge', Computers and Industrial Engineering, Vol 21(1-4), pp79-83

Jacobs, G (1994) 'The Division of Labour', Personal Computer Magazine, June, pp170-189

Kamburowski, J. (1985a) 'An Upper Bound on the Expected Completion Time of PERT Networks', European Journal of Operational Research, Vol 21(2), pp206-212

Kamburowski, J. (1985b) 'Bounds in Temporal Analysis of Stochastic Networks', Foundations of Control Engineering, Vol 10(4), pp177-189

Kelley, J.E. (1961) 'Critical Path Planning and Scheduling: Mathematical Basis', Operations Research, Vol 9, pp296-320

Kerzner, H. (1989) 'Project Management', Van Nostand Reinhold, New York

Kidd, J.B. (1990) 'Project Management Software - Are We Being Over-Persuaded?', International Journal of Project Management, Vol 8(2), pp109-115

Kidd, J.B. (1991) 'Do Today's Projects Need Powerful Network Planning Tools?', International Journal Production Research, Vol 29(10), pp1969-1978

King, W.R. (1964) 'A Stochastic Personnel Assignment Model', Operations Research, March, pp67-81

Kleindorfer, G.B. (1969) 'Bounding Distributions for Stochastic Activity Networks', Working Paper, School of Education, University of California, Berkely

Kunz, J.C. Bonura, T. and Stezlner, M.J. (1986) 'Contingent Analysis for Project Management Using Multiple Worlds', Applications of Artificial Intelligence in Engineering Problems, Vol 2, pp707-718

Lambourn, S. (1963) 'Resource Allocation and Multi-Project Scheduling (RAMPS) - A New Tool in Planning and Control', The Computer Journal, Vol 5(4), pp 300-304

Law, A.M. and Kelton, W.D. (1991) 'Simulation Modelling and Analysis', McGraw Hill Incorporated, New York, 2nd edition

Lee, S.E. Moeller, G.L. and Digman, L.A. (1982) 'Network Analysis for Management Decisions', Kluwer-Nijhoff Publishing, Boston

Lefèvre, C. (1986) 'Bounds for the Expectation of Linear Combinations of Order Statistics with Application to PERT Networks', Stochastic Analysis and Applications, Vol 4(3), pp351-356

Levine, H.A. (1986) 'Project Management Using Microcomputers', Osborne McGraw-Hill, California

Levitt, R. and Kartam, N. (1990) 'Expert Systems in Construction Engineering and Management: State of the Art', Knowledge Engineering Review, Vol 5(2), pp97-125

Levitt, R.E. and Kunz, J.C. (1987) 'Using Artificial Intelligence Techniques to Support Project Management', AI Edam, Vol 1(1), ©Academic Press Limited, pp3-24

Levy, F.K. Thompson, G.L. and Wiest, J.D. (1962) 'Multiship, Multistop Workload-Smoothing Program', Naval Research Logistics Quarterly, Vol 9(3), pp37-44

Levy, F.K. and Thompson, G.L. and Wiest, J.D. (1963) 'The ABCs of the Critical Path Method', Harvard Business Review, pp98-108

Liu, L. and Harowitz, E. (1989) 'A Formal Method for Software Project Management', IEEE Transactions on Software Engineering, Vol 15(10), pp1280-1293

Lootsma, F.A. (1989) 'Stochastic and Fuzzy PERT', European Journal of Operations Research, Vol 43, pp174-183

Lucey, T. (1987) 'Management Information Systems', 5th Edition, DP Publications Limited, Hampshire

MacCrimmon, K.R. and Ryavec, C.A. (1964) 'An Analytical Study of the PERT Assumptions', Operations Research, Vol 12, pp16-37

Macro, A. (1990) 'Software Engineering: Concepts and Management', Prentice Hall International (UK) Limited

Macro, A. and Buxton, J. (1987) 'The Craft of Software Engineering', Addison-Wesley Publishing Company, Wokingham

Malcolm, D.G. Roseboom, J.H. Clark, C.E. and Fazar, W. (1959) 'Application of a Technique for Research and Development Program Evaluation', Operations Research, Vol 7, pp646-669

Martin, J.J. (1965) 'Distribution of the Time Through a Directed, Acyclic Network', Operations Research, Vol 13, pp46-66

Martins, J. (1992) 'Belief Revision', in Shapiro, S. C. Ed, Encyclopedia of Artificial Intelligence, Vol 1, John Wiley and Sons Incorporated, New York, pp58-62

Martins, J.P. (1990) 'The Truth, The Whole Truth, and Nothing but the Truth: An Indexed Bibliography to the Literature of Truth Maintenance Systems', AI Magazine, Vol 11(5), pp7-25

Mazza, C. (1989) 'Software Project Management', Computer Physics Communications, Vol (57), pp23-28

McClellan, H.S. (1969) 'Bounds for use in stochastic network analysis', Master Thesis, Faculty of the School of Engineering, Air Force Instituite of Technology, Wright Patterson AFB, Ohio

McCracken, D.D. (1981) 'A Maverick Approach to Systems Analysis and Design', Systems Analysis and Design - A Foundation for the 1980s, Elsevier North Holland, pp551-553

McCracken, D.D. and Jackson, M.A. (1986) 'A Minority Dissenting Position', IEEE Computer Society Tutorial, EH0245-1, ISBN 0-8186-0707-6, p23

McDermott, D. and Doyle, J. (1979) 'An Introduction to Non-Monotonic Logic', IJCAI-79, 6th Conference, Tokyo, Vol 1, pp562-567

McDermott, D. and Doyle, J. (1980) 'Non-Monotonic Logic 1', Artificial Intelligence, Vol 13(1&2), pp41-72

McGowan, J.W. (1987) 'VERT-PC Placing a Powerful Analysis Tool at the Decisionmakers Fingertips', Proceedings of the of 1987 International Conference on Systems Man and Cybernetics, Alexandria, Virginia, USA, 20-23 October, Vol 1, pp274-280

McKay, M.D. Conover, W.J. and Beckman, R.J. (1979) 'A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code', Technometrics, Vol 211, pp239-245

Miller, R.W. (1962) 'How to Plan and Control with PERT', Harvard Business Review, Mar/Apr, pp93-104

Milton, R. (1994) 'A Game of Chance', Personal Computer Magazine, May pp198-209

Moder, J.J. and Phillips, C.R. (1983) 'Project Management with CPM and PERT', Reinhold Publishing Corporation, Chapman Hall Limited, London

Moeller, G.L. (1972) 'VERT - A Tool to Assess Risk', 23rd Conference of the American Institute of Industrial Engineering, pp211-221

Moeller, G.L. and Digman, L.A. (1981) 'Operations Planning with VERT', Operations Research, Vol 29(4), pp676-697

Mongalo, M.A. and Lee, J. (1990). 'A Comparative Study of Methods for Probabilistic Project Scheduling', Computers and Industrial Engineering, Vol 19(1-4), pp505-509

Moore, L.J. and Clayton, E.R. (1976) 'GERT Modelling and Simulation: Fundamentals and Applications', Petocelli/Charter, New York

Moore, L.J. and Taylor, B.W. (1977) 'Multiteam, Multiproject Research and Development Planning with GERT', Management Science Vol 24(4), pp401-410

Morreale, R. (1985) 'Project Planning and Control', Data Processing (GB), Vol 27(3), pp19-21

Neumann, K. (1984) 'Recent Developments in Stochastic Activity Networks', INFOR, Vol 22(3), August, pp219-248

Newell, A. (1962) 'Some Problems in Basic Organisation in Problem Solving Programs', in Yovits et alia (1962), pp393-423.

Nii, H.P. (1986a) 'Blackboard Systems: The Blackboard Model of Problem Solving and Evolution of Blackboard Architectures', AI Magazine, Summer, pp38-53

Nii, H.P. (1986b) 'Blackboard Systems, Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective', AI Magazine, August, pp82-106

Noronha, S.J. and Sarma, V.V.S. (1991) 'Knowledge-Based Approaches for Scheduling Problems: A Survey', IEEE Transactions on Knowledge and Data Engineering, Vol 3(2), pp160-171

Paige, H.W. (1963) 'How Pert-Cost Helps The General Manager', Harvard Busines Review, Nov/Dec, pp87-95

Peltu, M. (1994) 'Rising to the Challenge', Computing, 31 March, p47

Plasket, R.L. (1986) 'Project Management: New Technology Enhances Old Concepts', Journal of Systems Management, Vol 37(6), pp6-10

Platz, J. (1986) 'Project Management in the Development of Scientific Software', Computer Physics Communications, Vol 41(2-3), pp217-225

Pohl, J. and Chapman, A. (1987) 'Probabilistic Project Management', Building and Environment, Vol 22(3), pp209-214

Powell, M. (1990) 'Where Tools Meet Methods', Computer Weekly, 3 May, p26

Pressman, R.S. (1994) 'Software Engineering A Practitioner's Approach', 3rd Edition, European Adaption, McGraw Hill Incorporated, New York

Pritsker, A.A.B. (1974) 'The Precedence GERT User's Manual' Pritsker and Associates Incorporated, Lafayette

Pritsker A.A.B. (1979) 'Modelling and Analysis Using Q–GERT Networks', John Wiley and Sons Incorporated, New York

Pritsker, A.A.B. and Happ, W.W. (1966) 'GERT: Graphical Evaluation and Review Technique Part 1. Fundamentals' Journal of Industrial Engineering, Vol 17(5), pp267-274

Pritsker, A.A.B. and Sigal, C.E. (1983) 'Management Decision Making, A Network Simulation Approach', Prentice-Hall Incorporated, Englewood Cliffs, New Jersey

Pritsker, A.A.B. and Whitehouse, G.E. (1966) 'GERT: Graphical Evaluation and Review Technique Part II. Probabilistic and Industrial Engineering Applications', Journal of Industrial Engineering, Vol 17(6), pp293-301

Project Management South (1994) Novotel, Hammersmith, London, 22-23 March, Project Management Exhibitions Limited, Basingstoke, Hampshire

Pulk, B.E. (1990) 'Improving Software Project Management', Journal of Systems Software, Vol 13, pp231-235

Ragsdale, C. (1989) 'The Current State of Network Simulation in Project Management Theory and Practice', Omega International Journal of Management Science, Vol 17(1), pp21-25

Reiter, R. and DeKleer, J. (1987) 'Foundations of Assumption-Based Truth Maintenance Systems: Preliminary Report', Proceedings of the AAAI-87, 6th National Conference on Artificial Intelligence, Magan Kaufmann Publishers, Los Altos, pp183-188

Rexing, G.L. (1991) 'Software Project Management: Moving Beyond Project Plans', AT&T Technical Journal, Vol 70(2), pp40-48

Ringland, G.A. and Duce, D.A. (1988) 'Approaches to Knowledge Representation', Research Studies

Robillard, P. and Trahan, M. (1977) 'The completion Time of PERT Networks', Operations Research, Vol 25(1), pp15-29

Rodi, W.L. (1989) 'A New Algorithm for Truth Maintenance', The Annual AI systems in Government, Proceedings of the IEEE Conference, Washington, March, 0818619341, pp14-21

Rook, P. (1986) 'Controlling Software Projects', Software Engineering Journal, Vol 1(1), pp7-16

Royston, J.P. (1982) 'The W Test for Normality', Royal Statistical Society.

Rubinstein, R.Y. (1981) 'Simulation and the Monte Carlo Method', John Wiley and Sons Incorporated, New York

Saitow, A.R. (1969) 'CSPC: Reporting Project Progress to the Top', Harvard Busines Review, January/February, pp88-97

Samli, A.C. and Bellas, C. (1971) 'The Use of GERT in the Planning and Control of Marketing Research', Journal of Marketing Research, Vol 8, pp335-339

Sathi, A. Morton, T.E. and Roth, S.F. (1986) 'Callisto: An Intelligent Project Management System', The AI Magazine, Vol 5, pp34-52

Schach, S. R. (1993) 'Software Engineering', 2nd Edition, Aksen Associates Incorporated, Boston, USA

Schagen, I.P. (1986) 'Statistics and Operations Research', Chartwell-Bratt, Sweden

Schoderbek, P.P. and Digman, L.A. (1967) 'Third Generation, PERT/LOB', Harvard Busines Review, September/October, pp100-110

Schonberger, R.J. (1981) 'Why Projects are 'Always' Late: A Rationale Based on Manual Simulation of a PERT/CPM Network', Interfaces, Vol 11(5), pp66-70

Sculli, D. (1983) 'The Completion Time of PERT Networks', Journal of the Operational Research Society, Vol 34(2), pp155-158

Sculli, D. and Wong, K.L. (1985) 'The Maximum and Sum of Two Beta Variables and the Anlaysis of PERT Networks', Omega International Journal of Management Science, Vol 13(3), pp233-240

Shapiro, S.S. and Wilk, M.B. (1965) 'An Analysis of Variance Test for Normality (complete samples)', Biometrika, Vol 52(3&4), pp591-611

Shogan, A.W. (1977) 'Bounding Distributions for a Stochastic PERT Network', Networks, Vol 7(4), pp359-381

Shoham, Y. and McDermott, D.V. (1991) 'Temporal Reasoning', Encyclopedia of Artificial Intelligence, Ed. Shapiro, S. C., Vol 1, John Wiley and Sons, New York, pp1334-1339

Sigal, C.E. Pritsker, A.A.B. and Solberg, J.J. (1979) 'The Use of Cutsets in Monte Carlo Analysis of Stochastic Netwoks', Mathematics and Computers in Simulation, Vol 21, pp376-384

Sigal, C.E. Pritsker, A.A.B. and Solberg, J.J. (1980) 'The Stochastic Shortest Route Problem', Operations Research, Vol 28(5), pp579-593

Slagle, J.R. Gardiner, D.A. and Han, K. (1990) 'Knowledge Specification of an Expert System', IEEE Expert, August, pp29-38

Smith, B. (1985) 'Project Concepts', Effective Project Administration, Institute of Mechanical Engineers

Sodhi, J. (1991) 'Software Engineering Methods and Management and CASE Tools', TAB Professional and Reference Books, McGraw Hill Incorporated, Blue Ridge Summit, PA

Sommerville, I. (1993) 'Software Engineering', 4th Edition, Addison-Wesley Publishing Company, Wokingham

Spangler, W.E. (1991) 'The Role of Artificial Intelligence in Understanding the Strategic Decision-Making Process', IEEE Transactions on Knowledge and Data Engineering, Vol 3(2), pp149-159

Spelde, H.G. (1977) 'Bounds for the Distribution Function of Network Variables', Operations Research, Verfahren XXVII, pp113-123

Spikes Cavell (1993), Survey results, 8 April, Computing, p20

Stallman, R.M. and Sussman, G.J. (1977) 'Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis', Artificial Intelligence, Vol 9(2), pp135-159

Startzman, R.A. and Wattenbarger, R.A. (1985) 'An Improved Computation Procedure for Risk Analysis Problems with Unusual Probability Functions', SPE Hydrocarbon Economics and Evaluation Symposium Proceedings, Dallas

Sullivan, R.S. Hayya, J.C. and Schaul, R. (1982) 'Efficiency of the Antithetic Variate Method for Simulating Stochastic Networks', Management Science, Vol 28(5), pp563-572

Talbot, F.B. (1982) 'Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case', Management Science, Vol 28(10), pp1197-1210

Tausworthe, R.C. (1980) 'The Work Breakdown Structure in Software Project Management', Journal of Systems and Software, pp181-186

Taylor, B.W. and Moore, L.J. (1978) 'Project Management Using GERT Analysis', Project Management Quarterly, September, pp99-104

Todd, A. (1993) 'A Development Process', Computing, November 18, pp47-48

Tsang, E.P.K. (1988) 'Elements in Temporal Reasoning in Planning', Proceedings of 8th European Conference on AI, Munich, August 1-15, ISBN 0273087983, International Library, pp571-573

Tulip, A. (1983) 'Project Management Techniques Applied to Computing', Data Processing, Vol 25(7), pp9-27

Turner, J.R. (1993) 'The Handbook of Project-Based Management', McGraw-Hill Book Company Europe, UK

Urbanski, A. (1988) 'Formalizing Non-Monotonic Truth Maintenance Systems', Artificial Intelligence III: Methodology, Systems, Applications, pp43-50

Uyeno, D. (1992) 'Monte Carlo Simulation on Microcomputers', Simulation, Vol 58(6), pp418-423

Van Slyke, R.M. (1963) 'Monte Carlo Methods and the PERT Problem', Operations Research, Vol 11, pp839-860

Wetherill, G.B. (1981) 'Intermediate Statistical Methods', Chapman and Hall Limited, London

Whitehouse, G.E. (1973) 'Systems Analysis and Design Using Network Techniques', Prentice-Hall Incorporated, Englewood Cliffs, New Jersey

Whitehouse, G.E. and Pritsker, A.A.B. (1969) 'GERT: Part III - Further Statistical Results; Counters, Renewal Times and Correlations', AIIE Transactions, Vol 1(1), pp45-50

Wiest, J.D. (1967) 'A Heuristic Model for Scheduling Large Projects with Limited Resources', Management Science, Vol 13(3), ppB359-B377

Wiest, J.D. (1981) 'Precedence Diagramming Method: Some Unusual Characteristics and their Implications for Project Managers', Journal of Operations Management, Vol 1(3), pp213-222

Wilson, D.N. and Sifer, M.J. (1988) 'Structural Planning - Project Views', Software Engineering Journal, July, pp134-140

Wilson, D.N. and Sifer, M.J. (1990) 'Structured Planning: Deriving Project Views', Software Engineering Journal, March, pp138-148

Woodworth, B.M. and Willie, C.J. (1975) 'A Heuristic Algorithm for Resource Levelling in Multi-project, Multi-resource Scheduling', Decision Sciences, Vol 6, pp525-40

Yarnold, J.K. (1970) 'The Minimum Expectation in χ^2 Goodness-of-Fit Tests and the Accuracy of Approximations for the Null Distribution', Journal of the American Statistical Association, Vol 65, pp864-886

Yovits, M.C. Jacobi, G.T. and Goldstein, G.D. Eds. (1962) Conference on Self-Organising Systems, Spartan Books, Washington DC

SOFTWARE AND SUPPLIERS

@Risk™ Version 1.11 (1990) ©Palisade Corporation, New York

Artemis Prestige™ for Windows™ (1993) ©Lucas Management Systems, London

Artemis Schedule Publisher™ (1993) ©Advanced Management Solutions Incorporated

BestFit™ (1993) Distribution Fitting Software for Windows™, Release 1.0, ©Palisade Corporation, New York

CA-SuperProject® for Windows™ (1993) ©Computer Associates® International Incorporated, Islandia, New York

CS Project™ for Windows™ (1992) ©CREST Software, Leach Management Systems Ltd, Chippingham, Wiltshire

Cascade® Version 2.3.1 (1993) MANTIX Systems Limited, Bracknell, UK

Cobra® (1993) ©Welcom Software Technology International, London Road, London

InstaPlan™ (1990) ©Micro Planning International Limited, Bristol, Supplier Deepak Sareen Associates, Harrow-on-the-Hill, Middlesex

Methods On-line (1992), Learmonth and Burchett Management Systems (LBMS), London

MICRO PLANNER® Version 6 (1992) ©Micro Planning International Limited, Bristol

Microsoft® Project™ 4.0 (1994) ©Microsoft® Limited, Wokingham, Berkshire

Monte Carlo™ 2.0 (1993) ©Primavera Systems Incorporated, Bala Cynwyd, PA, USA

On Target™ (1991) ©Symantec (UK) Limited, Maidenhead, Berkshire

Open Plan® 4.0 (1993) ©Welcom Software Technology International, London Road, London

Opera® (1993) ©Welcom Software Technology International, London Road, London

Panorama™ COST for Windows™, Panorama Software Corporation Limited, W alton on Thames, Surrey

Panorama™ PLANNER for Windows™, Panorama Software Corporation Limited, Walton on Thames, Surrey

Parade® (1993) Primavera Systems Incorporated, Bala Cynwyd, PA, USA

PARISS Enterprise™ for Windows™ (1994) PARISS Limited, Notingham, UK

Pertmaster Advance 2.4G (1994) People in Technology Limited, St John Street, London

PLANTRAC-APROPOS®, Computerline Limited, Woodham, Weybridge, Surrey

PMSX-Kernel™ (1994) Aran Limited, Thatcham, Berks

PowerProject® Version 2 (1991) Asta Development Corporation Limited, Thame, Oxfordshire

Predict!™ (1992) ©Risk Decisions Limited, Oxford, UK

Primavera Project Planner® for Windows™ (1994) ©Primavera Systems Incorporated, Bala Cynwyd, PA, USA

Project Manager Workbench DOS and 3.1 for Windows™ (1983 and 1994), Hoskyns Group PLC, Wandsworth Road, London

Project RISK (1993) Hoskyns Group PLC, Wandsworth Road, London

Project Scheduler 6™ for Windows™ (1993), Scitor ® Corporation, Foster City, CA

Project/2 Series X®, ©Project Software and Development (PSDI) Incorporated (UK) Limited, Woking, Surrey

ProjectGuide™ (1993) Supplier Deepak Sareen Associates, Harrow-on-the-Hill, Middlesex

RISKNET™ (1992) ©Baesema, Broomielaw, Glasgow

Risnet™ (1993) JMCA (John Cockerham and Associates) Incorporated, Huntsville, Al, USA

Schedule Express™ (1993) Deepak Sareen Associates, Harrow-on-the-Hill, Middlesex

Schedule Publisher™ (1992) ©Project Managment Professional Services Limited, Maidenhead, Berkshire

SSADM Engineer (1992), Learmonth and Burchett Management Systems (LBMS), London

Texim Project™ 2.0 for Windows™ (1993) ©Welcom Software Technology International, London Road, London

Time Line® for Windows™ (1991), ©Symantec (UK) Limited, Maidenhead, Berkshire

TrackStar™, Complete Project Management Limied, Stokenchurch, Buckinghamshire