

BOB HUGHES AND MIKE COTTERELL

Software Project Management

Second Edition

www.mcgraw-hill.co.uk/hughes



Software Project Management

(Second Edition)

Software Project Management (Second Edition)

Bob Hughes and Mike Cotterell,
School of Information Management, University of Brighton

The McGraw-Hill Companies

London • Burr Ridge, IL • New York • St Louis • San Francisco • Auckland • Bogotá Caracas
Lisbon • Madrid • Mexico • Milan • Montreal • New Delhi • Panama • Paris • San Juan • São Paulo
Singapore • Tokyo • Toronto

Published by
McGraw-Hill Publishing Company
SHOPPENHANGERS ROAD, MAIDENHEAD, BERKSHIRE, SL6 2QL, ENGLAND
Telephone: +44(0) 1628 502500
Fax: +44(0) 1628 770224
Web site: <http://www.mcgraw-hill.co.uk>

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 007 709505 7

Library of Congress cataloguing in publication data
The LOC data for this book has been applied for and may be obtained from the Library of Congress,
Washington, D.C.

Authors' Web site address: <http://www.mcgraw-hill.co.uk/hughes>

Publishing Director: Alfred Waller
Publisher: David Hatter
Typesetting: Mouse Nous, Cross-in-Hand
Production: Steven Gardiner Ltd
Cover: Hybert Design



The McGraw-Hill Companies

Copyright © 1999 McGraw-Hill International (UK) Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise without the prior permission of McGraw-Hill International (UK) Limited.

While every precaution has been taken in the preparation of this book, neither the authors, nor McGraw-Hill, shall have any liability with respect to any loss or damage caused directly or indirectly by the instructions or advice contained in the book.

Printed in Great Britain at the University Press, Cambridge

1 2 3 4 5 CUP 3 2 1 0 9

The road to hell is paved with works-in-progress.

Philip Roth

Contents

1	Introduction to software project management	1
1.1	Introduction	1
1.2	What is a project?	2
1.3	Software projects versus other types of project	3
1.4	Activities covered by software project management	3
1.5	Some ways of categorizing software projects	6
1.6	The project as a system	7
1.7	What is management?	8
1.8	Problems with software projects	9
1.9	Management control	11
1.10	Stakeholders	13
1.11	Requirement specification	14
1.12	Information and control in organizations	15
1.13	Conclusion	17
1.14	Further exercises	18
2	Step Wise: an overview of project planning	19
2.1	Introduction to Step Wise project planning	19
2.2	Step 0: Select project	20
2.3	Step 1: Identify project scope and objectives	22
2.4	Step 2: Identify project infrastructure	24
2.5	Step 3: Analyse project characteristics	27
2.6	Step 4: Identify project products and activities	28
2.7	Step 5: Estimate effort for each activity	32
2.8	Step 6: Identify activity risks	32
2.9	Step 7: Allocate resources	33
2.10	Step 8: Review/publicize plan	34
2.11	Steps 9 and 10: Execute plan and Lower levels of planning	35
2.12	Conclusion	35
2.13	Further Exercises	36
3	Project evaluation	37
3.1	Introduction	37
3.2	Strategic assessment	38
3.3	Technical assessment	40

3.4	Cost–benefit analysis	40
3.5	Cash flow forecasting	42
3.6	Cost–benefit evaluation techniques	43
3.7	Risk evaluation	50
3.8	Conclusion	55
3.9	Further exercises	55
4	Selection of an appropriate project approach	57
4.1	Introduction	57
4.2	Choosing technologies	59
4.3	Technical plan contents list	63
4.4	Choice of process models	63
4.5	Structured methods	64
4.6	Rapid application development	64
4.7	The waterfall model	65
4.8	The V-process model	66
4.9	The spiral model	67
4.10	Software prototyping	67
4.11	Other ways of categorizing prototypes	70
4.12	Tools	71
4.13	A prototyping example	72
4.14	Incremental delivery	73
4.15	An incremental example	76
4.16	Selecting the most appropriate process model	76
4.17	Conclusion	77
4.18	Further exercises	77
5	Software effort estimation	79
5.1	Introduction	79
5.2	Where are estimates done?	81
5.3	Problems with over- and under-estimates	82
5.4	The basis for software estimating	84
5.5	Software effort estimation techniques	85
5.6	Expert judgement	87
5.7	Estimating by analogy	88
5.8	Albrecht function point analysis	89
5.9	Function points Mark II	92
5.10	Object points	94
5.11	A procedural code-oriented approach	96
5.12	COCOMO: a parametric model	97
5.13	Conclusion	103
5.14	Additional exercises	104

8.8 Publishing the resource schedule	162
8.9 Cost schedules	164
8.10 The scheduling sequence	165
8.11 Conclusion	167
8.12 Further exercises	167
9 Monitoring and control	169
9.1 Introduction	169
9.2 Creating the framework	169
9.3 Collecting the data	173
9.4 Visualizing progress	175
9.5 Cost monitoring	179
9.6 Earned Value	180
9.7 Prioritizing monitoring	185
9.8 Getting the project back to target	186
9.9 Change control	188
9.10 Conclusions	190
9.11 Further exercises	190
10 Managing contracts	191
10.1 Introduction	191
10.2 Types of contract	192
10.3 Stages in contract placement	198
10.4 Typical terms of a contract	203
10.5 Contract management	206
10.6 Acceptance	208
10.7 Summary	208
10.8 Further exercises	208
11 Managing people and organizing teams	211
11.1 Introduction	211
11.2 Understanding behaviour	213
11.3 Organizational behaviour: a background	213
11.4 Selecting the right person for the job	215
11.5 Instruction in the best methods	217
11.6 Motivation	217
11.7 Working in groups	221
11.8 Becoming a team	221
11.9 Decision making	224
11.10 Leadership	226
11.11 Organizational structures	229
11.12 Conclusion	232
11.13 Further exercises	232

8.8 Publishing the resource schedule	162
8.9 Cost schedules	164
8.10 The scheduling sequence	165
8.11 Conclusion	167
8.12 Further exercises	167
9 Monitoring and control	169
9.1 Introduction	169
9.2 Creating the framework	169
9.3 Collecting the data	173
9.4 Visualizing progress	175
9.5 Cost monitoring	179
9.6 Earned Value	180
9.7 Prioritizing monitoring	185
9.8 Getting the project back to target	186
9.9 Change control	188
9.10 Conclusions	190
9.11 Further exercises	190
10 Managing contracts	191
10.1 Introduction	191
10.2 Types of contract	192
10.3 Stages in contract placement	198
10.4 Typical terms of a contract	203
10.5 Contract management	206
10.6 Acceptance	208
10.7 Summary	208
10.8 Further exercises	208
11 Managing people and organizing teams	211
11.1 Introduction	211
11.2 Understanding behaviour	213
11.3 Organizational behaviour: a background	213
11.4 Selecting the right person for the job	215
11.5 Instruction in the best methods	217
11.6 Motivation	217
11.7 Working in groups	221
11.8 Becoming a team	221
11.9 Decision making	224
11.10 Leadership	226
11.11 Organizational structures	229
11.12 Conclusion	232
11.13 Further exercises	232

C.3	An overview of the EM acquisition process	292
C.4	Acquisition goal definition	292
C.5	Acquisition planning	293
C.6	Procurement	295
C.7	Adaptation planning	295
C.8	Method bridging	300
C.9	Conclusions	301
D	ISO 12207 – an overview	303
D.1	Introduction	303
D.2	The ISO 12207 approach to software life cycle data	303
D.3	The ISO 12207 approach to software life cycle processes	304
D.4	The acquisition process	305
D.5	The supply process	308
D.6	The development process	309
E	Project Management Bodies of Knowledge	315
E.1	Introduction	315
E.2	Project Management Institute	316
E.3	Australian Institute of Project Management	319
E.4	Association for Project Management	320
E.5	UK National Vocational Qualifications	322
E.6	Information Systems Examination Board	324
F	Answer pointers	327
	Further reading	367

Preface to the second edition

Since the first edition of *Software Project Management*, the perception of the importance of project management and consequently its development as a discipline has continued to grow. In the UK, for example, we have seen the publication of the British Standards Institution guidelines on project management (BS 6079) and a revised and improved version of the PRINCE standard. The British Computer Society professional examinations now include a paper on project management. The Association for Project Management has continued to develop its Body of Knowledge and its system of qualifications, while the Project Management Institute in the United States, through its seminal Body of Knowledge document, is making its influence felt world-wide. In a European context, Euromethod has been published. This initiative attempts, admittedly with mixed success, to address top-level project management issues. We have also been made aware of the impressive set of national vocational competence standards on project management that have been developed in Australia.

Our contacts with industry suggest that part of this growing interest in project management is because organizations have become 'leaner' and 'delayered', so that the burden of keeping businesses running has been put on the shoulders of a smaller and more hard-pressed work force. Staff who might regard themselves as primarily technical people often find that 'empowerment' means that they now have to plan and manage work where previously this would have been done for them. The removal of layers of management also means that organizational changes often require a project approach where previously they could have been implemented as part of normal day-to-day organizational management.

We have found some who have claimed that the managers of IT projects do not need to have any specific expertise in IT matters: that essentially there is no need for software project management. As the title of this book indicates, we are not of this view. As Darryl Ince of the Open University has noted, software disasters since 1995 have not abated and if anything have increased, especially where client-server software has been the subject of development. It seems clear that project managers need to be aware of the issues and problems of IT development and IT developers need to have project management skills.

The target audience for this book remains students of disciplines such as information technology, information systems and computer science where project management is part of their course; and also practitioners, typically IT developers who have just or are about to assume project management responsibilities.

Preface to the first edition

The effective management of projects in IT environments has increasingly been seen to be important in recent years. In the UK, some aspects of this have been:

- the development of the government-sponsored PRINCE standard
- the setting up of the PROMS-G project management special interest group of the British Computer Society
- the provision of a Certificate of Proficiency in project management by the Information Systems Examinations Board.

Our contacts with industry and commerce have underlined for us the significance of what might be regarded as very basic project management measures. It is our belief that these fundamental practices need to be stressed so that they become first nature for IT practitioners, as a very important part of the foundations of their professional education. While it is hoped that there may be something of interest in this book for the experienced project manager, it is targeted more directly at students about to enter the world of IT development, either through an industrial placement or a first job, and those already in software development who are just starting to take on project management responsibilities and who seek guidance.

Bearing in mind this audience, we have made extensive reference to two imaginary scenarios which explore the concerns of two new project leaders, Amanda and Brigette, who are undertaking their first project management roles.

We touch upon many techniques that may be of assistance when planning and controlling projects. Some of these may be more appropriate than others in particular circumstances. To give coherence we have provided Step Wise, a general framework compatible with PRINCE, for project planning into which the various techniques can be fitted.

We would especially like to thank Ken I'Anson of the CCTA for his helpful suggestions about the Appendix on PRINCE, and also Dave Hatter and Chris Clare for their guidance. The early advice of David Howe and Martin Campbell Kelly on the basic content and format of this book is also gratefully acknowledged. Thanks, too, to Barbara Kitchenham of the NCC, Manchester for her permission to us to use a project data set shown in the Chapter 5. We know that we have picked up many ideas from our colleagues and we acknowledge this particularly in the case of John Williams, Garth Glynn and Heinz Seefried. The work of putting

Chapter 1

Introduction to software project management

OBJECTIVES

When you have completed this chapter you will be able to:

- define the scope of 'software project management';
 - distinguish between software and other types of development project;
 - understand some problems and concerns of software project managers;
 - define the usual stages of a software project;
 - explain the main elements of the role of management;
 - understand the need for careful planning, monitoring and control;
 - identify the stakeholders of a project, their objectives and ways of measuring the success in meeting those objectives;
 - measure the success of a project in meeting its objectives.
-

1.1 Introduction

What exactly do we mean by 'software project management'? To answer this, we need to look at some key ideas about the planning, monitoring and control of software projects. Projects to produce software are worthwhile only if they satisfy real needs and so we will examine how we can identify the stakeholders in a project and their objectives. Identifying those objectives and checking that they are met is the basis of a successful project. This, however, cannot be done unless there is accurate information and how this is provided will be explored.

1.2 What is a project?

Dictionary definitions of 'project' include:

'A specific plan or design'
'A planned undertaking'
'A large undertaking: for example, a public works scheme'

Longman Concise English Dictionary, 1982.

The dictionary definitions put a clear emphasis on the project's being a planned activity.

Another key aspect of a project is that the undertaking is non-routine: a job which is repeated a number of times is not a project. There is a hazy boundary in between. The first time you do a routine task it will be very like a project. On the other hand, a project to develop a system that is very similar to previous ones that you have developed will have a large element of the routine.

We can summarize the key characteristics that distinguish projects as follows:

- non-routine tasks are involved;
- planning is required;
- specific objectives are to be met or a specified product is to be created;
- the project has a predetermined time span (which may be absolute or relative);
- work is carried out for someone other than yourself;
- work involves several specialisms;
- work is carried out in several phases;
- the resources that are available for use on the project are constrained;
- the project is large or complex.

In general, the more any of these factors applies to a task, the more difficult it is going to be to complete it successfully. Project size is particularly important. It should not be a surprise that a project that employs 200 project personnel is going to be rather trickier to manage than one that involves just two people. The examples and exercises used in this book usually relate to smaller projects. This is just to make them more manageable from a learning point of view: the techniques and issues discussed are of equal relevance to larger projects.

Exercise 1.1

Consider the following:

- producing an edition of a newspaper;
- building the Channel Tunnel;
- getting married;
- amending a financial computer system to deal with dates after the 31st December, 1999;
- a research project into what makes a good human-computer interface;
- an investigation into the reason why a user has a problem with a computer system;

- a programming assignment for a second year computing student;
- writing an operating system for a new computer;
- installing a new version of a word processing application in an organization.

Some would appear to merit the description ‘project’ more than others. Put them into an order that most closely matches your ideas of what constitutes a project. For each entry in the ordered list, describe the difference between it and the one above that makes it less worthy of the term ‘project’.

There is no one correct answer to this exercise, but a possible solution to this and the other exercises you will come across may be found at the end of the book.

1.3 Software projects versus other types of project

Many of the techniques of general project management are applicable to software project management, but Fred Brooks pointed out that the products of software projects have certain characteristics that make them different.

One way of perceiving software project management is as the process of making visible that which is invisible.

Invisibility When a physical artefact such as a bridge or road is being constructed the progress being made can actually be seen. With software, progress is not immediately visible.

Complexity Per dollar, pound or euro spent, software products contain more complexity than other engineered artefacts.

Flexibility The ease with which software can be changed is usually seen as one of its strengths. However this means that where the software system interfaces with a physical or organizational system, it is expected that, where necessary, the software will change to accommodate the other components rather than vice versa. This means the software systems are likely to be subject to a high degree of change.

Brooks, F. P. ‘No silver bullet: essence and accidents of software engineering’. This essay has been included in *The Mythical Man-Month*, Anniversary Edition, Addison-Wesley, 1995.

1.4 Activities covered by software project management

A software project is concerned not only with the actual writing of software. In fact, where a software application is bought in ‘off-the-shelf’, there might be no software writing as such. This is still fundamentally a software project because so many of the other elements associated with this type of project are present.

Chapter 4 on project analysis and technical planning looks at some alternative life cycles.

Usually, there are three successive processes that bring a new system into being:

1. **The feasibility study** This is an investigation to decide whether a prospective project is worth starting. Information will be gathered about the general requirements of the proposed system. The probable developmental

and operational costs, along with the value of the benefits of the new system are estimated. With a large system, the feasibility study could be treated as a project in its own right. This evaluation may be done as part of a strategic planning exercise where a whole range of potential software developments are evaluated and put into an order of priority. Sometimes an organization has a policy where a series of projects is planned as a *programme* of development.

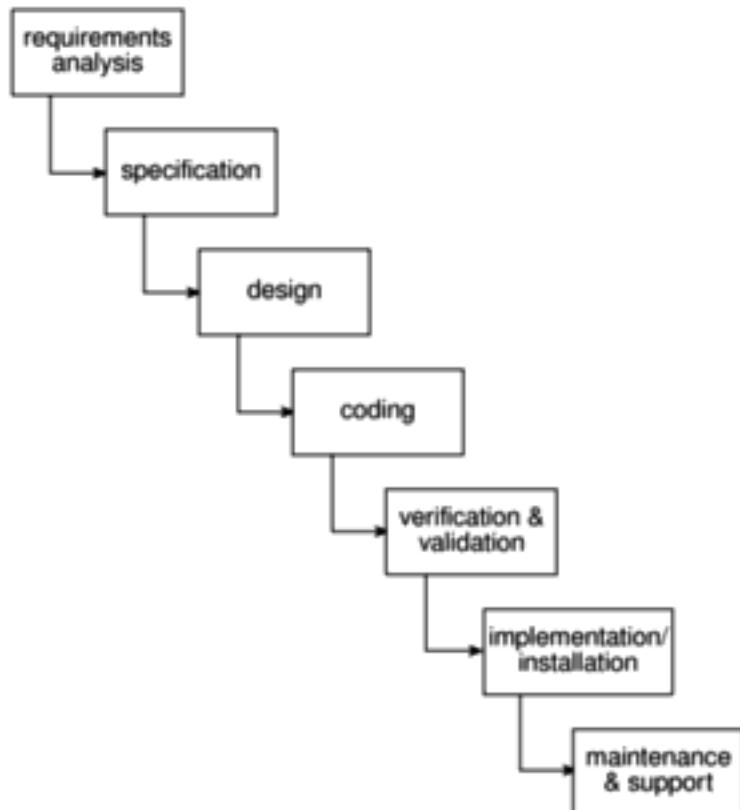


Figure 1.1 A typical project life-cycle.

The PRINCE 2 method, which is described in Appendix A takes this planning by stages approach.

2. **Planning** If the feasibility study produces results that indicate that the prospective project appears viable, then planning of the project can take place. In fact, for a large project, we would not do all our detailed planning right at the beginning. We would formulate an outline plan for the whole project and a detailed one for the first stage. More detailed planning of the later stages would be done as they approached. This is because we would have more detailed and accurate information upon which to base our plans nearer to the start of the later stages.
3. **Project execution** The project can now be executed. Individual projects are likely to differ considerably but a classic project life-cycle is shown in Figure 1.1.

The stages in the life-cycle illustrated in Figure 1.1 above are described in a little more detail below:

Requirements analysis This is finding out in detail what the users require of the system that the project is to implement. Some work along these lines will almost

certainly have been carried out when the project was evaluated but now the original information obtained needs to be updated and supplemented. Several different approaches to the users' requirements may be explored. For example, a small system that satisfies some, but not all, of the users' needs at a low price may be compared to a system with more functions but at a higher price.

Specification Detailed documentation of what the proposed system is to do.

Design A design that meets the specification has to be drawn up. This design activity will be in two stages. One will be the external or user design. This lays down what the system is to look like to the users in terms of menus, screen and report layouts and so on. The next stage produces the physical design, which tackles the way in which the data and software procedures are be structured internally.

Coding This might refer to writing code in a procedural language such as C or Ada, or might refer to the use of a high level application builder. Even where software is not being built from scratch, some modification to the base application might be required to meet the needs of the new application.

Verification and validation Whether software is developed specially for the current application or not, careful testing will be needed to check that the proposed system meets its requirements.

Implementation/installation Some system development practitioners refer to the whole of the project after design as 'implementation' (that is, the implementation of the design) while others insist that the term refers to the installation of the system after the software has been developed. In this case it encompasses such things as setting up data files and system parameters, writing user manuals and training users of the new system.

Maintenance and support Once the system has been implemented there will be a continuing need for the correction of any errors that may have crept into the system and for extensions and improvements to the system. Maintenance and support activities may be seen as a series of minor software projects. In many environments, most software development is in fact maintenance.

Brightmouth College is a higher education institution which used to be managed by the local government authority but has now become autonomous. Its payroll is still administered by the local authority and pay slips and other output are produced in the local authority's computer centre. The authority now charges the college for this service. The college management are of the opinion that it would be cheaper to obtain an 'off-the-shelf' payroll application and do the payroll processing themselves.

What would be the main stages of the project to convert to independent payroll processing by the college? Bearing in mind that an off-the-shelf application is to

Exercise 1.2

be used, how would this project differ from one where the software was to be written from scratch?

1.5 Some ways of categorizing software projects

It is important to distinguish between the main types of software project because what is appropriate in one context might not be so in another. For example, SSADM, the Structured Systems Analysis and Design Method, is suitable for developing information systems but not necessarily other types of system.

Information systems versus embedded systems

A distinction may be made between *information systems* and *embedded systems*. Very crudely, the difference is that in the former case the system interfaces with the organization, whereas in the latter case the system interfaces with a machine! A stock control system would be an information system that controls when the organization reorders stock. An embedded, or process control, system might control the air conditioning equipment in a building. Some systems may have elements of both so that the stock control system might also control an automated warehouse.

Embedded systems are also called real-time or industrial systems.

Exercise 1.3

Would an operating system on a computer be an information system or an embedded system?

Objectives versus products

Projects may be distinguished by whether their aim is to produce a *product* or to meet certain *objectives*.

A project might be to create a product the details of which have been specified by the client. The client has the responsibility for justifying the product.

On the other hand, the project might be required to meet certain objectives. There might be several ways of achieving these objectives in contrast to the constraints of the product-driven project. One example of this is where a new information system is implemented to improve some service to users inside or outside an organization. The subject of an agreement would be the level of service rather than the characteristics of a particular information system.

Many software projects have two stages. The first stage is an objectives-driven project, which results in a recommended course of action and may even specify a new software application to meet identified requirements. The next stage is a project actually to create the software product.

Service level agreements are becoming increasingly important as organizations contract out functions to external service suppliers.

Would the project to implement an independent payroll system at the Brightmouth College described in Exercise 1.2 above be an objectives-driven project or a product-driven project?

Exercise 1.4

1.6 The project as a system

A project is concerned with creating a new system and/or transforming an old one and is itself a system.

Systems, subsystems and environments

A simple definition of the term *system* is 'a set of interrelated parts'. A system will normally be part of a larger system and will itself comprise *subsystems*.

Outside the system there will be the system's *environment*. This will be made up of things that can affect the system but over which the system has no direct control. In the case of Brightmouth College, the bankruptcy of the main supplier of IT equipment would be an event happening in the system's environment.

Identify the possible subsystems of the installed Brightmouth College payroll system.

Exercise 1.5

What important entities exist in the payroll system's environment?

Open versus closed systems

Open systems are those that interact with the environment. Nearly all systems are open. One reason that engineered systems and the projects to construct them often fail is that the technical staff involved do not appreciate the extent to which systems are open and are liable to be affected by outside changes.

Sub-optimization

This is where a subsystem is working at its optimum but is having a detrimental effect on the overall system. An example of this might be where software developers deliver to the users a system that is very efficient in its use of machine resources, but is also very difficult to modify.

Sociotechnical systems

Software projects belong to this category of systems. Any software project requires both technological organization and also the organization of people. Software project managers therefore need to have both technical competence and the ability to interact persuasively with other people.

1.7 What is management?

The Open University suggest that management involves the following activities:

- planning – deciding what is to be done;
- organizing – making arrangements;
- staffing – selecting the right people for the job, for example;
- directing – giving instructions;
- monitoring – checking on progress;
- controlling – taking action to remedy hold-ups;
- innovating – coming up with new solutions;
- representing – liaising with users etc.

Exercise 1.6

Paul Duggan is the manager of a software development section. On Tuesday at 10.00 am he and his fellow section heads have a meeting with their group manager about the staffing requirements for the coming year. Paul has already drafted a document 'bidding' for staff. This is based on the work planned for his section for the next year. The document is discussed at the meeting. At 2.00 pm Paul has a meeting with his senior staff about an important project his section is undertaking. One of the software development staff has just had a road accident and will be in hospital for some time. It is decided that the project can be kept on schedule by transferring another team member from less urgent work to this project. A temporary replacement is to be brought in to do the less urgent work but this might take a week or so to arrange. Paul has to phone both the personnel manager about getting a replacement and the user for whom the less urgent work is being done explaining why it is likely to be delayed.

Identify which of the eight management responsibilities listed above Paul was responding to at different points during his day.

Another way of looking at the management task is to ask managers what their most frequent challenges are. A survey of software project managers produced the following list:

- coping with deadlines (85%);
- coping with resource constraints (83%);
- communicating effectively among task groups (80%);
- gaining commitment from team members (74%);
- establishing measurable milestones (70%);
- coping with changes (60%);

The results of this survey by H. J. Thamhain and D. L. Wilemon appeared in June 1986 in *Project Management Journal* under the title 'Criteria for controlling software according to plan'.

- working out project plan agreement with their team (57%);
- gaining commitment from management (45%);
- dealing with conflict (42%);
- managing vendors and sub-contractors (38%).

Similar lists appear in the computer trade press, for example in the 27 August 1998 edition of *Computing*.

The percentages relate to the numbers of managers identifying each challenge. A manager could identify more than one.

1.8 Problems with software projects

One way of deciding what ought to be covered in 'software project management' is to consider what problems need to be addressed.

Traditionally, management has been seen as the preserve of a distinct class within the organization. As technology has made the tasks undertaken by an organization more sophisticated, many management tasks seem to have become dispersed throughout the organization: there are management systems rather than managers. Nevertheless, the successful project will normally have one person who is responsible for its success. Such people are likely to be concerned with the key areas that are most likely to prevent success – they are primarily trouble-shooters and their job is likely to be moulded by the problems that confront the project. A survey of managers published by Thayer, Pyster and Wood identified the following commonly experienced problems:

- poor estimates and plans;
- lack of quality standards and measures;
- lack of guidance about making organizational decisions;
- lack of techniques to make progress visible;
- poor role definition – who does what?
- incorrect success criteria.

Further details of the survey can be found in 'Major issues in software engineering project management' in *IEEE Transactions on Software Engineering*, Volume 7, pp 333–342.

The above list looks at the project from the manager's point of view. What about the staff who make up the members of the project team? Below is a list of the problems identified by a number of students on a degree course in Computing and Information Systems who had just completed a year's industrial placement:

- inadequate specification of work;
- management ignorance of IT;
- lack of knowledge of application area;
- lack of standards;
- lack of up-to-date documentation;

- preceding activities not completed on time – including late delivery of equipment;
- lack of communication between users and technicians;
- lack of communication leading to duplication of work;
- lack of commitment – especially when a project is tied to one person who then moves;
- narrow scope of technical expertise;
- changing statutory requirements;
- changing software environment;
- deadline pressure;
- lack of quality control;
- remote management;
- lack of training.

Note how many of the problems identified by the students stemmed from poor communications. Another common problem identified by this and other groups of students is the wide range of IT specialisms – an organization may be made up of lots of individuals or groups who will be expert in one set of software techniques and tools but ignorant of those used by their colleagues. Communication problems are therefore bound to arise.

Stephen Flower's *Software Failure, Management Failure*, Wiley & Sons, 1996, is an interesting survey of failed computer projects

What about the problems faced by the customers of the products of computer projects? Here are some recent stories in the press:

- the United States Internal Revenue System was to abandon its tax system modernization programme after having spent \$4 billion;
- the state of California spent \$1 billion on its non-functional welfare database system;
- the £339 million United Kingdom air traffic control system was reported as being two years behind schedule;
- a discount stock brokerage company had 50 people working 14 hours or more a day to correct three months of records clerically—the report commented that the new system had been rushed into operation without adequate testing;
- in the United Kingdom, a Home Office immigration service computerization project was reported as having missed two deadlines and was nine months late;
- the Public Accounts Committee of the House of Commons in the United Kingdom blamed software bugs and management errors for £12 million of project costs in relation to an implementation of a Ministry of Agriculture computer system to administer farm subsidies.

Most of the stories above relate to public sector organizations. This may be misleading—private sector organizations tend to conceal their disasters and in any case many of the public projects above were actually being carried out by private sector contractors. Any lingering faith by users in the innate ability of IT people to plan ahead properly will have been removed by consideration of the ‘millennium bug’, a purely self-inflicted IT problem. On balance it might be a good idea not to survey users about their problems with IT projects!

1.9 Management control

The project control cycle

Management, in general, can be seen as the process of setting objectives for a system and then monitoring the system to see what its true performance is. In Figure 1.2 the ‘real world’ is shown as being rather formless. Especially in the case of large undertakings, there will be a lot going on about which management should be aware. As an example, take an IT project that is to replace locally held paper-based records with a centrally-organized database. It might be that staff in a large number of offices that are geographically dispersed need training and then need to use the new IT system to set up the back-log of manual records on the new database. It might be that the system cannot be properly operational until the last record has been transferred. It might also be the case that the new system will be successful only if new transactions can be processed within certain time cycles. The managers of the project ought to be asking questions about such things as how effective training has been, how many records have still to be transferred to the new database and transfer rates. This will involve the local managers in *data collection*. Bare details, such as ‘location X has processed 2000 documents’ will not be very useful to higher management: *data processing* will be needed to transform this raw *data* into useful *information*. This might be in such forms as ‘percentage of records processed’, ‘average documents processed per day per person’ and ‘estimated completion date’.

In the example above, the project leader might examine the ‘estimated completion date’ for completing data transfer for each branch and compare this with the overall target date for completion of this phase of the project. In effect they are comparing actual performance with one aspect of the overall project objectives. They might find that one or two branches are not going to complete the transfer of details in time, and would then need to consider what to do (this is represented in Figure 1.2 by the box *making decisions/plans*). One possibility would be to move staff temporarily from one branch to another. If this is done, there is always the danger that while the completion date for the one branch is pulled back to before the overall target date, the date for the branch from which staff are being moved is pushed forward beyond that date. The project manager would need to calculate carefully what the impact would be in moving staff from particular branches. This is *modelling* the consequences of a potential solution.

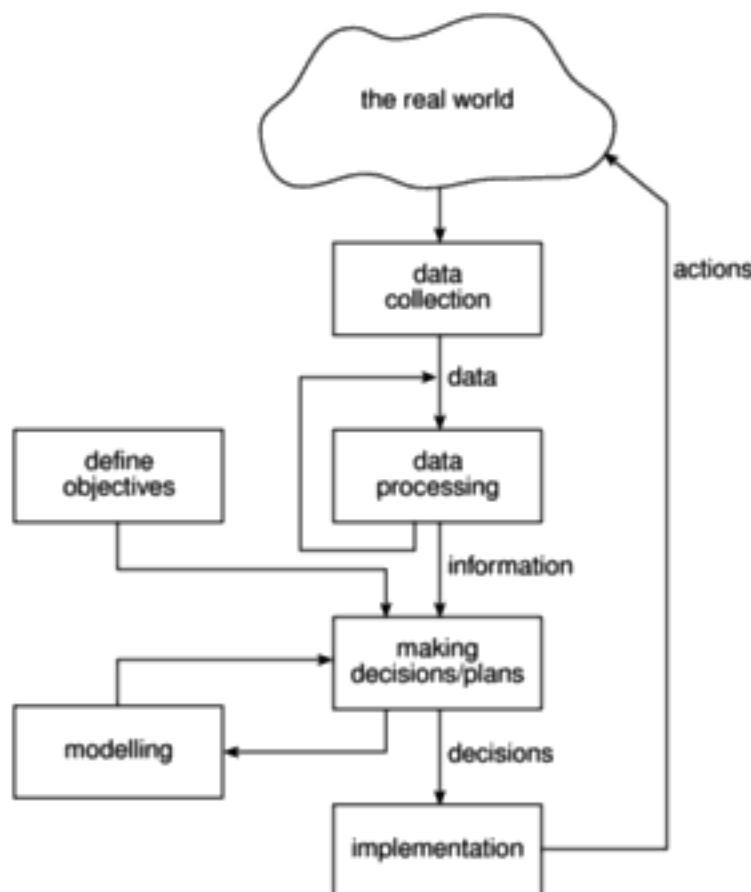


Figure 1.2 The project control cycle.

Several different proposals could be modelled in this way before one was chosen for *implementation*.

Having implemented the decision, the situation needs to be kept under review by collecting and processing further progress details. For instance, the next time that progress is reported, a branch to which staff have been transferred might still be behind in transferring details. This might be because the reason why the branch has got behind in transferring details is because the manual records are incomplete and another department, for whom the project has a low priority, has to be involved in providing the missing information. In this case, transferring extra staff to do data input will not have accelerated data transfer.

Objectives

Project objectives should be clearly defined.

To have a successful software project, the manager and the project team members must know what will constitute success. This will make them concentrate on what is essential to project success.

There might be more than one set of users of a system and there might be different groups of staff who are involved in its development. There is a need for well defined objectives that are accepted by all these people. Where there is more than one user group, then a *project authority* needs to be identified. Such a project authority has overall authority over what the project is to achieve.

This authority is often held by a *project steering committee*, which has overall responsibility for setting, monitoring and modifying objectives. The project manager still has responsibility for running the project on a day to day basis, but has to report to the steering committee at regular intervals. Only the steering committee can authorize changes to the project objectives and resources.

This committee is likely to contain user, development and management representatives.

Measures of effectiveness

Effective objectives are concrete and well defined. Vague aspirations such as 'to improve customer relations' are unsatisfactory. Objectives should be such that it is obvious to all whether the project has been successful or not. Ideally there should be *measures of effectiveness*, which tell us how successful the project has been. For example, 'to reduce customer complaints by 50%' would be more satisfactory as an objective than 'to improve customer relations'.

The measure can, in some cases, be an answer to simple yes/no question, 'Did we install the new software by 1st June?' for example.

Sub-objectives and goals

In order to keep things manageable, objectives might need to be broken down into sub-objectives. Here we say that in order to achieve A we must achieve B, C and D first. These sub-objectives are also known as *goals*, steps on the way to achieving an objective, just as goals scored in a football match are steps towards the objective of winning the match.

Identify the objectives and sub-objectives of the Brightmouth College payroll project. What measures of effectiveness could be used to check the success in achieving the objectives of the project?

Exercise 1.7

1.10 Stakeholders

These are people who have a stake or interest in the project. It is important that they be identified as early as possible, because you need to set up adequate communication channels with them right from the start. The project leader also has to be aware that not everybody who is involved with a project has the same motivation and objectives. The end users might, for instance, be concerned about the ease of use of the system while their managers might be interested in the staff savings the new system will allow.

Stakeholders might be internal to the project team, external to the project team but in the same organization, or totally external to the organization.

- **Internal to the project team** This means that they will be under the direct managerial control of the project leader.
- **External to the project team but within the same organization** For example, the project leader might need the assistance of the information

management group in order to add some additional data types to a database or the assistance of the users to carry out systems testing. Here the commitment of the people involved has to be negotiated.

- **External to both the project team and the organization** External stakeholders might be customers (or users) who will benefit from the system that the project implements or contractors who will carry out work for the project. One feature of the relationship with these people is that it is likely to be based on a legally binding contract.

Within each of the general categories there will be various groups. For example, there will be different types of user with different types of interests.

Different types of stakeholder might have different objectives and one of the jobs of the successful project leader is to recognize these different interests and to be able to reconcile them. It should therefore come as no surprise that the project leader needs to be a good communicator and negotiator. Boehm and Ross proposed a 'Theory W' of software project management where the manager concentrates on creating situations where all parties involved in a project benefit from it and therefore have an interest in its success. (The 'W' stands for Everyone a Winner.)

B. W. Boehm and R. Ross
'Theory W Software
Project Management:
Principles and Examples',
in B. W. Boehm (ed.)
*Software Risk
Management*.

Exercise 1.8

Identify the stakeholders in the Brightmouth College payroll project.

1.11 Requirement specification

Very often, especially in the case of product-driven projects, the objectives of the project are carefully defined in terms of functional requirements, quality requirements, and resource requirements.

- **Functional requirements** These define what the system that will be the end product of the project is to do. Systems analysis and design methods, such as SADT and Information Engineering, are designed primarily to provide functional requirements.
- **Quality requirements** There will be other attributes of the system to be implemented that do not relate so much to what the system is to do but how it is to do it. These are still things that the user will be able to experience. They include, for example, response time, the ease of using the system and its reliability.
- **Resource requirements** A record of how much the organization is willing to spend on the system. There will usually be a trade-off between this and the time it takes to implement the system. In general it costs disproportionately more to implement a system by an earlier date than a later one. There might

These are sometimes called non-functional requirements.

also be a trade-off between the functional and quality requirements and cost. We would all like exceptionally reliable and user-friendly systems that do exactly what we want but we might not be able to afford them.

1.12 Information and control in organizations

Hierarchical information and control systems

With small projects, the project leaders are likely to be working very closely with the other team members and might even be carrying out many non-managerial tasks themselves. Therefore they should have a pretty good idea of what is going on. When projects are larger, many separate teams will be working on different aspects of the project and the overall managers of the project are not going to have day-to-day direct contact with all aspects of the work.

Larger projects are likely to have a *hierarchical* management structure (Figure 1.3). Project team members will each have a group leader who allocates them work and to whom they report progress. In turn the group leader, along with several other group leaders, will report to a manager at the next higher level. That manager might have to report to another manager at a higher level, and so on.

There might be problems that cannot be resolved at a particular level. For example, additional resources might be needed for some task, or there might be a disagreement with another group. These will be referred to the next higher level of management.

At each higher level more information will be received by fewer people. There is thus a very real danger that managers at the higher levels might be overloaded with too much information. To avoid this, at each level the information will have to be summarized.

The larger the project, the bigger the communication problems.

The referral of disagreements to a higher level is sometimes known as *escalation*.

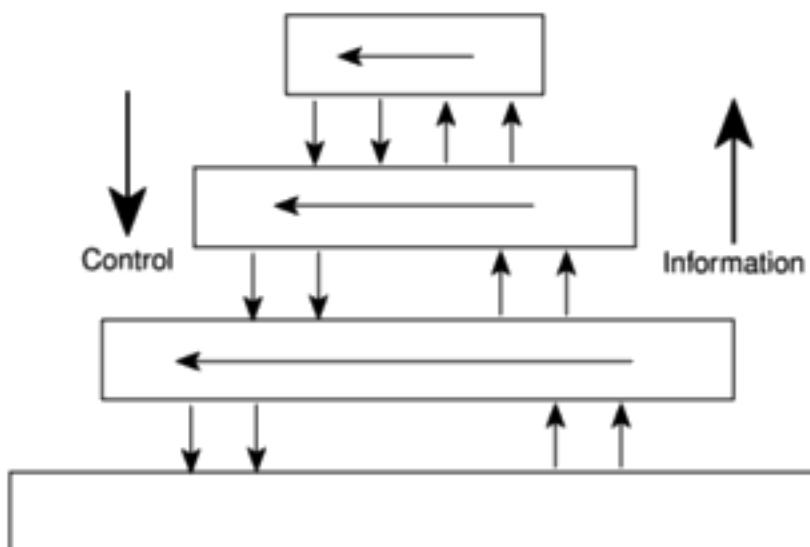


Figure 1.3 Management information flows up the organizational structure and control flows down.

As a result of examining the progress information and comparing it against what was planned, some remedial action might need to be taken. Instructions may be formulated and passed down to a lower level of management. The lower level managers will have to interpret what needs to be done and formulate more detailed plans to fulfil the directive. As the directives filter down the hierarchy, they will be expanded into more detail at each level.

For example, a programmer will receive a specification from an analyst and might then seek clarification.

Not all information flows concerning a project will be going up and down the hierarchy. There will also be lateral flows between groups and individuals on the same level.

Levels of decision making and information

Each decision made in a project environment should be based on adequate information of the correct sort. The type of information needed depends on the level of decision making. Decisions can be grouped at three levels: *strategic*, *tactical*, and *operational*.

Strategic decision making is essentially about deciding objectives. In the case of the Brightmouth College payroll, the decision to become administratively independent could be regarded as a strategic decision. In our example we were interested only in the payroll, but this might have been part of a wider programme which may have affected many other administrative functions.

Tactical decision making is needed to ensure that the objectives will be fulfilled. The project leader who has the responsibility for achieving objectives will have to formulate a plan of action to meet those objectives. The project leader will need to monitor progress to see whether these objectives are likely to be met and to take action where needed to ensure that the things remain on course.

Operational decisions relate to the day-to-day work of implementing the project. Deciding the content of the acceptance tests might come under this heading.

Differences in types of information

Table 1.1 gives some idea of the differences in the kind of information needed. There is a kind of continuum for most of the qualities suggested and what is needed for tactical decision making comes somewhere in the middle.

Effectiveness is concerned with doing the right thing. *Efficiency* is carrying out a task making the best possible use of the resources.

Measurement

The leader of a small project will have direct contact with many aspects of the project. With larger projects, project leaders would have to depend on information being supplied to them. This information should not be vague and ideally should be quantitative. This ties in with our need for unambiguous measures of effectiveness.

Software development deals largely with intangibles and does not easily lend itself to quantitative measures, but attempts are increasingly being made to introduce measurement into the software process.

The quantification of measures of effectiveness reduces ambiguity.

Table 1.1 *The types of information required for decision making*

<i>Characteristic</i>	<i>Operational</i>	<i>Strategic</i>
motivation	efficiency	effectiveness
orientation	internal	internal and external
focus	specific to a function	specific to organization
detail	detailed	summarized
response	fast	not so fast
access paths	standard	flexible
up-to-dateness	essential	desirable
accuracy	essential	approximate
certainty	essential	often predictive
objectivity	high	more subjective
information type	mainly quantitative	often qualitative

Software measurements can be divided into *performance measures* and *predictive measures*.

- **Performance measures** These measure the characteristics of a system that has been delivered. They are important when we are trying to specify unambiguously the quality requirements of a proposed system.
- **Predictive measures** The trouble with performance measures is that you need to have a system actually up and running before you can take measurements. As a project leader, what you want to be able to do is to get some idea of the likely characteristics of the final system during its development. *Predictive measures* are taken during development and indicate what the performance of the final system is likely to be.

Performance measures include mean time between failures (reliability) and time to learn an application (usability).

For example, the errors found per KLOC (that is, thousand lines of code) at different stages of the project might help to predict the correctness and reliability of the final system. Keystrokes required to carry out a particular transaction might help predict what the operator time to carry out the transaction is likely to be. Modularity, the degree to which the software is composed of self-contained manageable components, helps predict how easy changes to the final system will be.

1.13 Conclusion

This chapter has laid a foundation for the remainder of the book by defining what is meant by various terms such as 'software project' and 'management'. Among some of the more important points that have been made are the following.

- Projects are by definition non-routine and therefore more uncertain than normal undertakings.

Chapter 3

Project evaluation

OBJECTIVES

When you have completed this chapter you will be able to:

- carry out an evaluation and selection of projects against strategic, technical and economic criteria;
 - use a variety of cost-benefit evaluation techniques for choosing among competing project proposals;
 - evaluate the risk involved in a project and select appropriate strategies for minimizing potential costs.
-

3.1 Introduction

Deciding whether or not to go ahead with a project is really a case of comparing a proposed project with the alternatives and deciding whether to proceed with it. That evaluation will be based on strategic, technical and economic criteria and will normally be undertaken as part of strategic planning or a feasibility study for any information system development. The risks involved also need to be evaluated.

In this chapter we shall be using the term *project* in a broader sense than elsewhere in the book. Our decision as to whether or not to proceed with a project needs to be based upon whether or not it is desirable to carry out the development and *operation* of a software system. The term project may therefore be used, in this context, to describe the whole life cycle of a system from conception through to final decommissioning.

Project evaluation is normally carried out in Step 0 of Step Wise (Figure 3.1). The subsequent steps of Step Wise are then concerned with managing the development project that stems from this project selection.

'Do nothing' is an option which should always be considered.

The BS 6079 guidelines (see Appendix B) use the term project in this way.

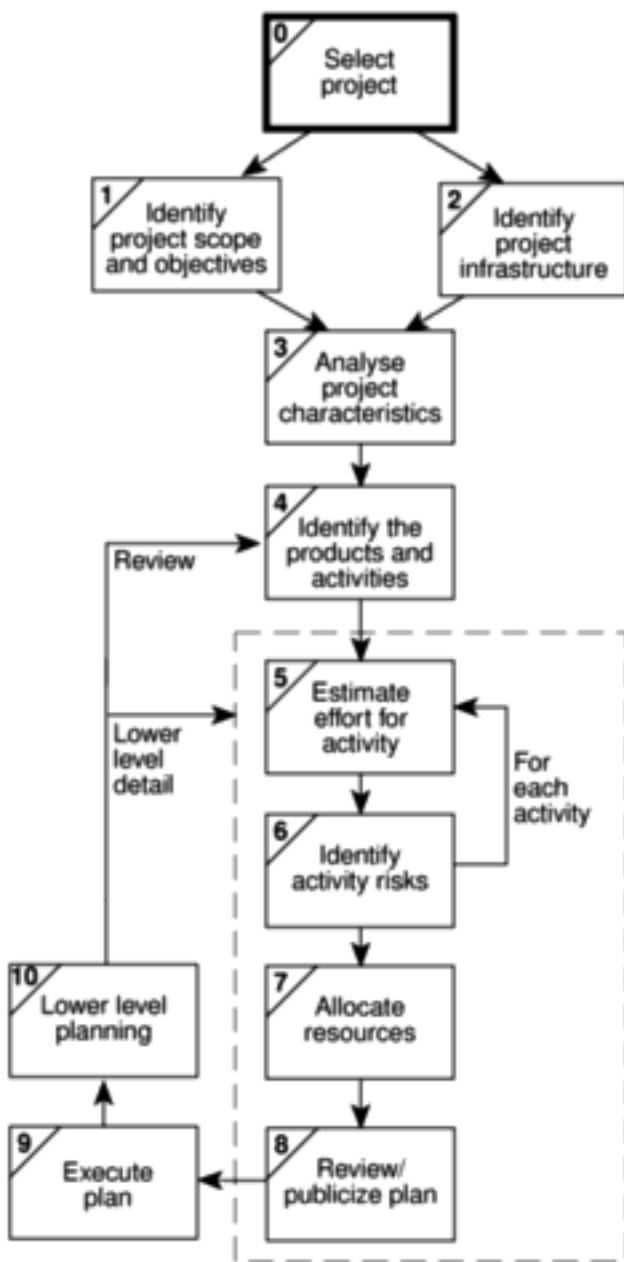


Figure 3.1 Project evaluation is carried out in Step 0.

D. C. Ferns defined a programme as 'a group of projects that are managed in a co-ordinated way to gain benefits that would not be possible were the projects to be managed independently' in a seminal article in the *International Journal of Project Management*, August 1991.

3.2 Strategic assessment

Programme management

It is being increasingly recognized that individual projects need to be seen as components of a *programme* and should be evaluated and managed as such. A programme, in this context, is a collection of projects that all contribute to the same overall organizational goals. Effective programme management requires that there is a well defined *programme goal* and that all the organization's projects are selected and tuned to contribute to this goal. A project must be evaluated according to how it contributes to this programme goal and its viability, timing, resourcing and final worth can be affected by the programme as a whole. It is to be expected

that the value of any project is increased by the fact that it is part of a programme – the whole, as they say, being greater than the sum of the parts.

In order to carry out a successful strategic assessment of a potential project there should therefore be a strategic plan clearly defining the organization's objectives. This provides the context for defining the programme and programme goals and, hence, the context for assessing the individual project. It is likely, particularly in a large organization, that there will be an organizational structure for programme management and it will be, for example, the *programme director* and *programme executive*, rather than, say, a project manager, who will be responsible for the strategic assessment of a proposed project.

Even where there is no explicitly defined programme, any proposed project must be evaluated within the context of the organization's overall business objectives. Moreover, any potential software system will form part of the user organization's overall information system and must be evaluated within the context of the existing information system and the organization's information strategy. Table 3.1 illustrates typical issues that must be addressed as part of the strategic assessment of a project.

Table 3.1 *Typical issues and questions to be considered during strategic assessment*

<i>Issue</i>	<i>Typical questions</i>
Objectives	How will the proposed system contribute to the organization's stated objectives? How, for example, might it contribute to an increase in market share?
IS plan	How does the proposed system fit into the IS plan? Which existing system(s) will it replace/interface with? How will it interact with systems proposed for later development?
Organization structure	What effect will the new system have on the existing departmental and organization structure? Will, for example, a new sales order processing system overlap existing sales and stock control functions?
MIS	What information will the system provide and at what levels in the organization? In what ways will it complement or enhance existing management information systems?
Personnel	In what way will the proposed system affect manning levels and the existing employee skill base? What are the implications for the organization's overall policy on staff development?
Image	What, if any, will be the effect on customers' attitudes towards the organization? Will the adoption of, say, automated systems conflict with the objectives of providing a friendly service?

Where a well-defined information systems strategy does not exist, system development and the assessment of project proposals will be based on a more piecemeal approach – each project being individually assessed early in its life cycle. In such cases it is likely that cost–benefit analysis will have more importance and some of the questions of Table 3.1 will be more difficult to answer.

Portfolio management

Third party developers must also carry out strategic and operational assessment of project proposals.

Where an organization such as a software house is developing a software system they could be asked to carry out a strategic and operational assessment on behalf of the customer. Whether or not this should be the case, they will require an assessment of any proposed project themselves. They will need to ensure that carrying out the development of a system is consistent with their own strategic plan – it is unlikely, for example, that a software house specializing in financial and accounting systems would wish to undertake development of a factory control system unless their strategic plan placed an emphasis on diversification.

The proposed project will form part of a *portfolio* of ongoing and planned projects and the selection of projects must take account of the possible effects on other projects in the portfolio (competition for resources, for example) and the overall portfolio profile (for example, specialization versus diversification).

3.3 Technical assessment

Technical assessment of a proposed system consists of evaluating the required functionality against the hardware and software available. Where an organization has a strategic information systems plan, this is likely to place limitations on the nature of solutions that might be considered. The constraints will, of course, influence the cost of the solution and this must be taken into account in the cost–benefit analysis.

3.4 Cost–benefit analysis

The most common way of carrying out an economic assessment of a proposed information system, or other development, is by comparing the expected costs of development and operation of the system with the benefits of having it in place.

Assessment is based upon the question of whether the estimated costs are exceeded by the estimated income and other benefits. Additionally, it is usually necessary to ask whether or not the project under consideration is the best of a number of options. There might be more candidate projects than can be undertaken at any one time and, in any case, projects will need to be prioritized so that any scarce resources may be allocated effectively.

The standard way of evaluating the economic benefits of any project is to carry out a cost–benefit analysis, which consists of two steps.

Any project requiring an investment must, as a minimum, provide a greater benefit than putting that investment in, say, a bank.

- **Identifying and estimating all of the costs and benefits of carrying out the project** This includes development costs of the system, the operating costs and the benefits that are expected to accrue from the operation of the system. Where the proposed system is replacing an existing one, these estimates should reflect the costs and benefits due to the new system. A sales order processing system, for example, could not claim to benefit an organization by the total value of sales – only by the increase due to the use of the new system.
- **Expressing these costs and benefits in common units** We must evaluate the net benefit, which is the difference between the total benefit and the total cost. To do this, we must express each cost and each benefit in monetary terms.

Most costs are relatively easy to identify and quantify in approximate monetary terms. It is helpful to categorize costs according to where they originate in the life of the project.

Many costs are easy to identify and measure in monetary terms.

- **Development costs** – include the salaries and other employment costs of the staff involved in the development project and all associated costs.
- **Setup costs** – include the costs of putting the system into place. These consist mainly of the costs of any new hardware and ancillary equipment but will also include costs of file conversion, recruitment and staff training.
- **Operational costs** – consist of the costs of operating the system once it has been installed.

Benefits, on the other hand, are often quite difficult to quantify in monetary terms even once they have been identified. Benefits may be categorized as follows.

- **Direct benefits** – these accrue directly from the operation of the proposed system. These could, for example, include the reduction in salary bills through the introduction of a new, computerized system.
- **Assessable indirect benefits** – these are generally secondary benefits, such as increased accuracy through the introduction of a more user-friendly screen design where we might be able to estimate the reduction in errors, and hence costs, of the proposed system.
- **Intangible benefits** – these are generally longer term or benefits that are considered very difficult to quantify. Enhanced job interest can lead to reduced staff turnover and, hence, lower recruitment costs.

Indirect benefits, which are difficult to estimate, are sometimes known as intangible benefits.

Brightmouth College are considering the replacement of the existing payroll service, operated by a third party, with a tailored, off-the-shelf computer-based system. List some of the costs and benefits they might consider under each of the six headings given above. For each cost or benefit, explain how, in principle, it might be measured in monetary terms.

Exercise 3.1

If a proposal shows an excess of benefits over costs then it is a candidate for further consideration.

Any project that shows an excess of benefits over costs is clearly worth considering for implementation. However, as we shall see later, it is not a sufficient justification for going ahead: we might not be able to afford the costs; there might be even better projects we could allocate our resources to instead; the project might be too risky.

3.5 Cash flow forecasting

As important as estimating the overall costs and benefits of a project is the forecasting of the cash flows that will take place and their timing. A cash flow forecast will indicate when expenditure and income will take place (Figure 3.2).

Typically products generate a negative cash flow during their development followed by a positive cash flow over their operating life. There might be decommissioning costs at the end of a product's life

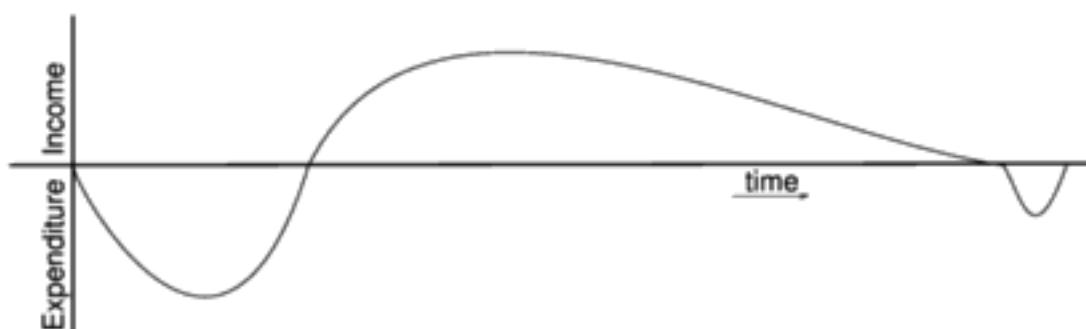


Figure 3.2 Typical product life cycle cash flow.

We need to spend money, such as staff wages, during the development stages of a project. Such expenditure cannot be deferred until income is received (either from using the software if it is being developed for in-house use or from selling it). It is important that we know that we can fund the development expenditure either from the company's own resources or by borrowing from the bank. In any event, it is vital to have some forecast of when expenditure such as the payment of salaries and bank interest will take place and when any income is to be expected, such as payment on completion or, possibly, stage payments.

Accurate cash flow forecasting is not easy, as it generally needs to be done early in the project's life cycle (at least before any significant expenditure is committed) and many items to be estimated (particularly the benefits of using software or decommissioning costs) might be some years in the future.

When estimating future cash flows, it is usual to ignore the effects of inflation. Trying to forecast the effects of inflation increases the uncertainty of the forecasts. Moreover, if expenditure is increased due to inflation it is likely that income will increase proportionately. However, measures to deal with increases in costs where work is being done for an external customer must be in place – for example index linked prices where work involves use of raw materials – see Chapter 10 on contract management.

Table 3.2 illustrates cash flow forecasts for four projects. In each case it is assumed that the cash flows take place at the end of each year. For short-term projects or where candidate projects demonstrate significant seasonal cash flow

The difficulty and importance of cash flow forecasting is evidenced by the number of companies that suffer bankruptcy because, although they are developing profitable products or services, they cannot sustain an unplanned negative cash flow.

patterns it can be advisable to produce quarterly, or even monthly, cash flow forecasts.

3.6 Cost–benefit evaluation techniques

We would consider proceeding with a project only where the benefits outweigh the costs. However, in order to choose among projects, we need to take into account the timing of the costs and benefits as well as the benefits relative to the size of the investment.

Consider the project cash flow estimates for four projects at IOE shown in Table 3.2. Negative values represent expenditure and positive values income.

Exercise 3.2

Rank the four projects in order of financial desirability and make a note of your reasons for ranking them in that way before reading further.

In the following sections we will take a brief look at some common methods for comparing projects on the basis of their cash flow forecasts.

Net profit

The net profit of a project is the difference between the total costs and the total income over the life of the project. Project 2 in Table 3.2 shows the greatest net profit but this is at the expense of a large investment. Indeed, if we had £1m to invest, we might undertake all of the other three projects and obtain an even greater net profit. Note also, that all projects contain an element of risk and we might not be prepared to risk £1m. We shall look at the effects of risk and investment later in this chapter.

Table 3.2 Four project cash flow projections – figures are end of year totals (£)

Year	Project 1	Project 2	Project 3	Project 4
0	-100,000	-1,000,000	-100,000	-120,000
1	10,000	200,000	30,000	30,000
2	10,000	200,000	30,000	30,000
3	10,000	200,000	30,000	30,000
4	20,000	200,000	30,000	30,000
5	100,000	300,000	30,000	75,000
Net profit	50,000	100,000	50,000	75,000

Cash flows take place at the end of each year. The year 0 figure represents the initial investment made at the start of the project.

Moreover, the simple net profit takes no account of the timing of the cash flows. Projects 1 and 3 each have a net profit of £50,000 and therefore, according to this selection criterion, would be equally preferable. The bulk of the income occurs late in the life of project 1, whereas project 3 returns a steady income throughout

its life. Having to wait for a return has the disadvantage that the investment must be funded for longer. Add to that the fact that, other things being equal, estimates in the more distant future are less reliable than short-term estimates and we can see that the two projects are not equally preferable.

Payback period

The *payback period* is the time taken to break even or pay back the initial investment. Normally, the project with the shortest payback period will be chosen on the basis that an organization will wish to minimize the time that a project is 'in debt'.

Exercise 3.3

Consider the four project cash flows given in Table 3.2 and calculate the payback period for each of them.

The advantage of the payback period is that it is simple to calculate and is not particularly sensitive to small forecasting errors. Its disadvantage as a selection technique is that it ignores the overall profitability of the project – in fact, it totally ignores any income (or expenditure) once the project has broken even. Thus the fact that projects 2 and 4 are, overall, more profitable than project 3 is ignored.

Return on investment

The *return on investment* (ROI), also known as the *accounting rate of return* (ARR), provides a way of comparing the net profitability to the investment required. There are some variations on the formula used to calculate the return on investment but a straightforward common version is

$$\text{ROI} = \frac{\text{average annual profit}}{\text{total investment}} \times 100$$

Exercise 3.4

Calculating the ROI for project 1, the net profit is £50,000 and the total investment is £100,000. The return on investment is therefore calculated as

$$\begin{aligned}\text{ROI} &= \frac{\text{average annual profit}}{\text{total investment}} \times 100 \\ &= \frac{10,000}{100,000} \times 100 = 10\%\end{aligned}$$

Calculate the ROI for each of the other projects shown in Table 3.2 and decide which, on the basis of this criterion, is the most worthwhile.

The return on investment provides a simple, easy to calculate measure of return on capital and is therefore quite popular. Unfortunately it suffers from two severe disadvantages. Like the net profitability, it takes no account of the timing of the cash flows. More importantly, it is tempting to compare the rate of return with current interest rates. However, this rate of return bears no relationship to the interest rates offered or charged by banks (or any other normal interest rate) since it takes no account of the timing of the cash flows or of the compounding of interest. It is therefore, potentially, very misleading.

Net present value

The calculation of *net present value* is a project evaluation technique that takes into account the profitability of a project and the timing of the cash flows that are produced. It does so by discounting future cash flows by a percentage known as the discount rate. This is based on the view that receiving £100 today is better than having to wait until next year to receive it, because the £100 next year is worth less than £100 now. We could, for example, invest the £100 in a bank today and have £100 plus the interest in a year's time. If we say that the present value of £100 in a year's time is £91, we mean that £100 in a year's time is the equivalent of £91 now.

The equivalence of £91 now and £100 in a year's time means we are discounting the future income by approximately 10% – that is, we would need an extra 10% to make it worthwhile waiting for a year. An alternative way of considering the equivalence of the two is to consider that, if we received £91 now and invested for a year at an annual interest rate of 10%, it would be worth £100 in a year's time. The annual rate by which we discount future earnings is known as the *discount rate* – 10% in the above example.

Similarly, £100 received in 2 years' time would have a present value of approximately £83 – in other words, £83 invested at an interest rate of 10% would yield approximately £100 in 2 years' time.

The present value of any future cash flow may be obtained by applying the following formula

$$\text{present value} = \frac{\text{value in year } t}{(1 + r)^t}$$

where r is the discount rate, expressed as a decimal value and t is the number of years into the future that the cash flow occurs.

Alternatively, and rather more easily, the present value of a cash flow may be calculated by multiplying the cash flow by the appropriate discount factor. A small table of discount factors is given in Table 3.3.

The NPV for a project is obtained by discounting each cash flow (both negative and positive) and summing the discounted values. It is normally assumed that any initial investment takes place immediately (indicated as year 0) and is not discounted. Later cash flows are normally assumed to take place at the end of each year and are discounted by the appropriate amount.

Net present value (NPV) and internal rate of return (IRR) are collectively known as discounted cash flow (DCF) techniques.

Note that this example uses approximate figures – when you have finished reading this section you should be able to calculate the exact figures yourself.

Table 3.3 Table of NPV discount factors

Year	Discount rate (%)					
	5	6	8	10	12	15
1	0.9524	0.9434	0.9259	0.9091	0.8929	0.8696
2	0.9070	0.8900	0.8573	0.8264	0.7972	0.7561
3	0.8638	0.8396	0.7938	0.7513	0.7118	0.6575
4	0.8227	0.7921	0.7350	0.6830	0.6355	0.5718
discount factor = $\frac{1}{(1+r)^t}$	0.7835	0.7473	0.6806	0.6209	0.5674	0.4972
for various values of r (the discount rate) and t (the number of years from now)	0.7462	0.7050	0.6302	0.5645	0.5066	0.4323
7	0.7107	0.6651	0.5835	0.5132	0.4523	0.3759
8	0.6768	0.6274	0.5403	0.4665	0.4039	0.3269
9	0.6446	0.5919	0.5002	0.4241	0.3606	0.2843
10	0.6139	0.5584	0.4632	0.3855	0.3220	0.2472
15	0.4810	0.4173	0.3152	0.2394	0.1827	0.1229
20	0.3769	0.3118	0.2145	0.1486	0.1037	0.0611
25	0.2953	0.2330	0.1460	0.0923	0.0588	0.0304

Exercise 3.5

Assuming a 10% discount rate, the NPV for project 1 (Table 3.2) would be calculated as in Table 3.4. The net present value for Project 1, using a 10% discount rate is therefore £618. Using a 10% discount rate, calculate the net present values for projects 2, 3 and 4 and decide which, on the basis of this, is the most beneficial to pursue.

Table 3.4 Applying the discount factors to project 1

Year	Project 1 cash flow (£)	Discount factor @ 10%	Discounted cash flow (£)
0	-100,000	1.0000	-100,000
1	10,000	0.9091	9,091
2	10,000	0.8264	8,264
3	10,000	0.7513	7,513
4	20,000	0.6830	13,660
5	100,000	0.6209	62,090
Net Profit:	£50,000		NPV: £618

It is interesting to note that the net present values for projects 1 and 3 are significantly different – even though they both yield the same net profit and both have the same return on investment. The difference in NPV reflects the fact that, with project 1, we must wait longer for the bulk of the income.

The main difficulty with NPV for deciding between projects is selecting an appropriate discount rate. Some organizations have a standard rate but, where this is not the case, then the discount rate should be chosen to reflect available interest rates (borrowing costs where the project must be funded from loans) plus some premium to reflect the fact that software projects are inherently more risky than lending money to a bank. The exact discount rate is normally less important than ensuring that the same discount rate is used for all projects being compared. However, it is important to check that the ranking of projects is not sensitive to small changes in the discount rate – have a look at the following exercise.

Calculate the net present value for each of the projects A, B and C shown in Table 3.5 using each of the discount rates 8%, 10% and 12%.

For each of the discount rates, decide which is the best project. What can you conclude from these results?

Exercise 3.6

Alternatively, the discount rate can be thought of as a target rate of return. If, for example, we set a target rate of return of 15% we would reject any project that did not display a positive net present value using a 15% discount rate. Any project that displayed a positive NPV would be considered for selection – perhaps by using an additional set of criteria where candidate projects were competing for resources.

Table 3.5 Three estimated project cash flows

Year	Project A (£)	Project B (£)	Project C (£)
0	– 8,000	– 8,000	– 10,000
1	4,000	1,000	2,000
2	4,000	2,000	2,000
3	2,000	4,000	6,000
4	1,000	3,000	2,000
5	500	9,000	2,000
6	500	–6,000	2,000
Net Profit	4,000	5,000	6,000

Internal rate of return

One disadvantage of NPV as a measure of profitability is that, although it may be used to compare projects, it might not be directly comparable with earnings from other investments or the costs of borrowing capital. Such costs are usually quoted

as a percentage interest rate. The internal rate of return (IRR) attempts to provide a profitability measure as a percentage return that is directly comparable with interest rates. Thus, a project that showed an estimated IRR of 10% would be worthwhile if the capital could be borrowed for less than 10% or if the capital could not be invested elsewhere for a return greater than 10%.

The IRR is calculated as that percentage discount rate that would produce an NPV of zero. It is most easily calculated using a spreadsheet or other computer program that provides functions for calculating the IRR. Microsoft Excel and Lotus, for example, both provide IRR functions which, provided with an initial guess or seed value (which may be zero), will search for and return an IRR.

Manually, it must be calculated by trial-and-error or estimated using the technique illustrated in Figure 3.3. This technique consists of guessing two values

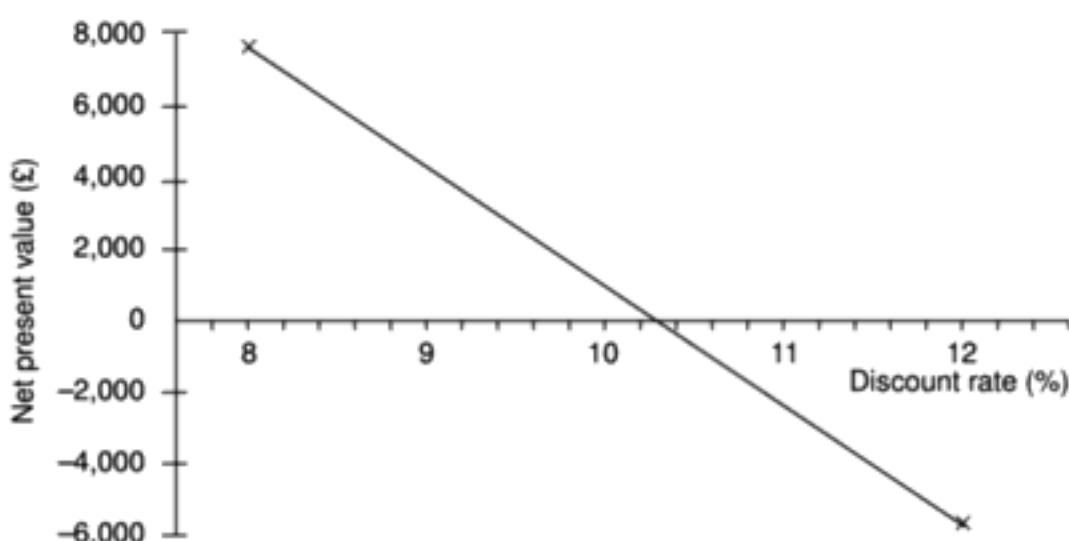


Figure 3.3 Estimating the internal rate of return for project 1.

(one either side of the true value) and using the resulting NPVs (one of which must be positive and the other negative) to estimate the correct value. Note that this technique will provide only an approximate value but, in many cases that can be sufficient to dismiss a project that has a small IRR or indicate that it is worth making a more precise evaluation.

The internal rate of return is a convenient and useful measure of the value of a project in that it is a single percentage figure that may be directly compared with rates of return on other projects or interest rates quoted elsewhere.

Table 3.6 illustrates the way in which a project with an IRR of 10% may be directly compared with other interest rates. The cash flow for the project is shown in column (a). Columns (b) to (e) show that if we were to invest £100,000 now at an annual interest rate of 10% in, say, a bank, we could withdraw the same amounts as we would earn from the project at the end of each year, column (e), and we would be left with a net balance of zero at the end. In other words, investing in a project that has an IRR of 10% can produce exactly the same cash flow as lending the money to a bank at a 10% interest rate. We can therefore reason

that a project with an IRR greater than current interest rates will provide a better rate of return than lending the investment to a bank. We can also say that it will be worth borrowing to finance the project if it has an IRR greater than the interest rate charged on the loan.

Table 3.6 A project cash flow treated as an investment at 10%

Year	(a) Project cash flow forecast (£)	Equivalent investment at 10%				£100,000 invested at 10% may be used to generate the cash flows shown. At the end of the 5-year period the capital and the interest payments will be entirely consumed leaving a net balance of zero.
		(b) Capital at start of year (£)	(c) Interest during year (£)	(d) Capital at end of year (£)	(e) End of year withdrawal (£)	
0	-100,000	—	—	—	—	
1	10,000	100,000	10,000	110,000	10,000	
2	10,000	100,000	10,000	110,000	10,000	
3	10,000	100,000	10,000	110,000	10,000	
4	20,000	100,000	10,000	110,000	20,000	
5	99,000	90,000	9,000	99,000	99,000	
6		0	0	0	0	

One deficiency of the IRR is that it does not indicate the absolute size of the return. A project with an NPV of £100,000 and an IRR of 15% can be more attractive than one with an NPV of £10,000 and an IRR of 18% – the return on capital is lower but the net benefits greater.

An often quoted objection to the internal rate of return is that, under certain conditions, it is possible to find more than one rate that will produce a zero NPV. This is not a valid objection since, if there are multiple solutions, it is always appropriate to take the lowest value and ignore the others. Spreadsheets will normally always return the lowest value if provided with zero as a seed value.

NPV and IRR are not, however, a complete answer to economic project evaluation.

- A total evaluation must also take into account the problems of funding the cash flows – will we, for example, be able to repay the interest on any borrowed money and pay development staff salaries at the appropriate time?
- While a project's IRR might indicate a profitable project, future earnings from a project might be far less reliable than earnings from, say, investing with a bank. To take account of the risk inherent in investing in a project, we might require that a project earn a 'risk premium' (that is, it must earn, say, at least 15% more than current interest rates) or we might undertake a more detailed risk analysis as described in the following sections of this chapter.
- We must also consider any one project within the financial and economic framework of the organization as a whole – if we fund this one, will we also be able to fund other worthy projects?

3.7 Risk evaluation

There is a risk that software might exceed the original specification and that a project will be completed early and under budget. That is not a risk that need concern us.

Every project involves risk of some form. When assessing and planning a project, we are concerned with the risk of the project's not meeting its objectives. In Chapter 8 we shall discuss ways of analysing and minimizing risk during the development of a software system. In this chapter, we are concerned with taking risk into account when deciding whether to proceed with a proposed project.

Risk identification and ranking

In any project evaluation we should attempt to identify the risks and quantify their potential effects. One common approach to risk analysis is to construct a project risk matrix utilizing a checklist of possible risks and to classify each risk according to its relative importance and likelihood. Note that the importance and likelihood need to be separately assessed – we might be less concerned with something that, although serious, is very unlikely to occur than with something less serious that is almost certain. Table 3.7 illustrates a basic project risk matrix listing some of the risks that might be considered for a project, with their importance and likelihood classified as high (H), medium (M), low (L) or exceedingly unlikely (—). So that projects may be compared the list of risks must be the same for each project being assessed. It is likely, in reality, that it would be somewhat longer than shown and more precisely defined.

The project risk matrix may be used as a way of evaluating projects (those with high risks being less favoured) or as a means of identifying and ranking the risks for a specific project. In Chapter 7 we shall consider a method for scoring the importance and likelihood of risks that may be used in conjunction with the project risk matrix to score and rank projects.

Risk and net present value

Where a project is relatively risky it is common practice to use a higher discount rate to calculate net present value. This addition or risk premium, might, for example, be an additional 2% for a reasonably safe project or 5% for a fairly risky one. Projects may be categorized as high, medium or low risk using a scoring method and risk premiums designated for each category. The premiums, even if arbitrary, provide a consistent method of taking risk into account.

Cost–benefit analysis

A rather more sophisticated approach to the evaluation of risk is to consider each possible outcome and estimate the probability of its occurring and the corresponding value of the outcome. Rather than a single cash flow forecast for a project, we will then have a set of cash flow forecasts, each with an associated probability of occurring. The value of the project is then obtained by summing the cost or benefit for each possible outcome weighted by its corresponding probability. Exercise 3.7 illustrates how this may be done.

Table 3.7 A fragment of a basic project risk matrix

<i>Risk</i>	<i>Importance</i>	<i>Likelihood</i>
Software never completed or delivered	H	—
Project cancelled after design stage	H	—
Software delivered late	M	M
Development budget exceeded $\leq 20\%$	L	M
Development budget exceeded $> 20\%$	M	L
Maintenance costs higher than estimated	L	L
Response time targets not met	L	H

BuyRight, a software house, is considering developing a payroll application for use in academic institutions and is currently engaged in a cost-benefit analysis. Study of the market has shown that, if they can target it efficiently and no competing products become available, they will obtain a high level of sales generating an annual income of £800,000. They estimate that there is a 1 in 10 chance of this happening. However, a competitor might launch a competing application before their own launch date and then sales might generate only £100,000 per year. They estimate that there is a 30% chance of this happening. The most likely outcome, they believe, is somewhere in between these two extremes – they will gain a market lead by launching before any competing product becomes available and achieve an annual income of £650,000. BuyRight have therefore calculated their expected sales income as in Table 3.8.

Total development costs are estimated at £750,000 and sales are expected to be maintained at a reasonably constant level for at least four years. Annual costs of marketing and product maintenance are estimated at £200,000, irrespective of the market share gained. Would you advise them to go ahead with the project?

This approach is frequently used in the evaluation of large projects such as the building of new motorways, where variables such as future traffic volumes, and hence the total benefit of shorter journey times, are subject to uncertainty. The technique does, of course, rely on our being able to assign probabilities of occurrence to each scenario and, without extensive study, this can be difficult.

When used to evaluate a single project, the cost–benefit approach, by ‘averaging out’ the effects of the different scenarios, does not take account an organization’s reluctance to risk damaging outcomes. Because of this, where overall profitability is the primary concern, it is often considered more appropriate for the evaluation of a portfolio of projects.

Risk profile analysis

An approach which attempts to overcome some of the objections to cost–benefit averaging is the construction of risk profiles using sensitivity analysis.

Exercise 3.7

Table 3.8 *BuyRight's income forecasts*

Sales	Annual sales income (£) <i>i</i>	Probability <i>p</i>	Expected Value (£) <i>i × p</i>
High	800,000	0.1	80,000
Medium	650,000	0.6	390,000
Low	100,000	0.3	30,000
Expected Income			500,000

This involves varying each of the parameters that affect the project's cost or benefits to ascertain how sensitive the project's profitability is to each factor. We might, for example, vary one of our original estimates by plus or minus 5% and recalculate the expected costs and benefits for the project. By repeating this exercise for each of our estimates in turn we can evaluate the sensitivity of the project to each factor.

By studying the results of a sensitivity analysis we can identify those factors that are most important to the success of the project. We then need to decide whether we can exercise greater control over them or otherwise mitigate their effects. If neither is the case, then we must live with the risk or abandon the project.

Sensitivity analysis demands that we vary each factor one at a time. It does not easily allow us to consider the effects of combinations of circumstances, neither does it evaluate the chances of a particular outcome occurring. In order to do this we need to use a more sophisticated tool such as Monte Carlo simulation. There are a number of risk analysis applications available (such as *@Risk* from Palisade) that use Monte Carlo simulation and produce risk profiles of the type shown in Figure 3.4.

Projects may be compared as in Figure 3.4, which compares three projects with the same expected profitability. Project A is unlikely to depart far from this expected value compared to project B, which exhibits a larger variance. Both of these have symmetric profiles, which contrast with project C. Project C has a skewed distribution, which indicates that although it is unlikely ever to be much more profitable than expected, it is quite likely to be far worse.

Using decision trees

The approaches to risk analysis discussed previously rather assume that we are passive bystanders allowing nature to take its own course – the best we can do is to reject over-risky projects or choose those with the best risk profile. There are many situations, however, where we can evaluate whether a risk is important and, if it is, indicate a suitable course of action.

Many such decisions will limit or affect future options and, at any point, it is important to be able to see into the future to assess how a decision will affect the future profitability of the project.

Prior to giving Amanda the job of extending their invoicing system, IOE must consider the alternative of completely replacing the existing system – which they

For an explanation of the Monte Carlo technique see any textbook on operational research.

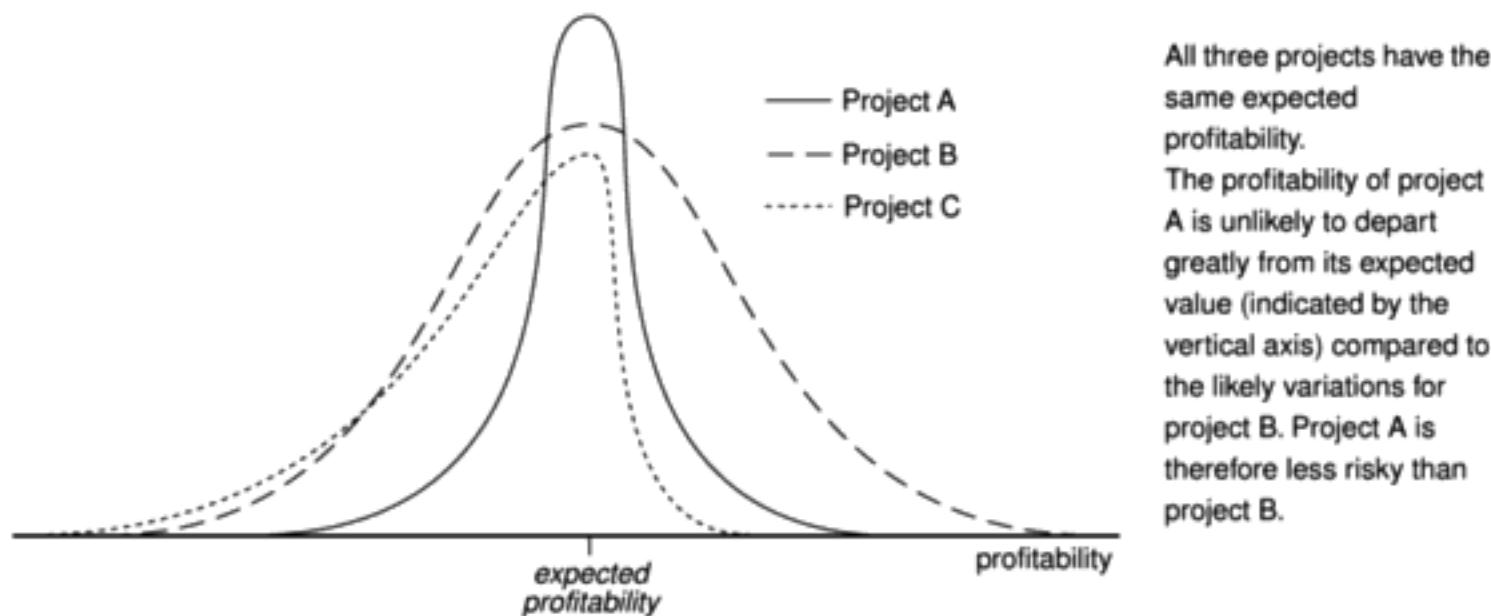


Figure 3.4 A risk analysis profile.

will have to do at some point in the future. The decision largely rests upon the rate at which their equipment maintenance business expands – if their market share significantly increases (which they believe will happen if rumours of a competitor's imminent bankruptcy are fulfilled) the existing system might need to be replaced within 2 years. Not replacing the system in time could be an expensive option as it could lead to lost revenue if they cannot cope with the increase in invoicing demand. Replacing it immediately will, however, be expensive as it will mean deferring other projects that have already been scheduled.

They have calculated that extending the existing system will have an NPV of £57,000, although if the market expands significantly, this will be turned into a loss with an NPV of -£100,000 due to lost revenue. If the market does expand, replacing the system now has an NPV of £250,000 due to the benefits of being able to handle increased sales and other benefits such as improved management information. If sales do not increase, however, the benefits will be severely reduced and the project will suffer a loss with an NPV of -£50,000.

The company estimate the likelihood of the market increasing significantly at 20% – and, hence, the probability that it will not increase as 80%. This scenario can be represented as a tree structure as shown in Figure 3.5.

The analysis of a decision tree consists of evaluating the expected benefit of taking each path from a decision point (denoted by D in Figure 3.5). The expected value of each path is the sum of the value of each possible outcome multiplied by its probability of occurrence. The expected value of extending the system is therefore £40,000 ($75,000 \times 0.8 - 100,000 \times 0.2$) and the expected value of replacing the system £10,000 ($250,000 \times 0.2 - 50,000 \times 0.8$). IOE should therefore choose the option of extending the existing system.

This example illustrates the use of a decision tree to evaluate a simple decision at the start of a project. One of the great advantages of using decision trees to

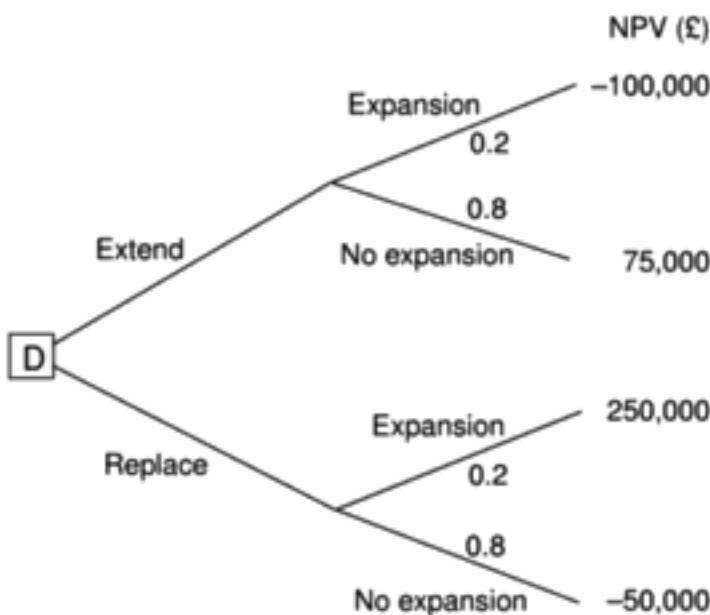
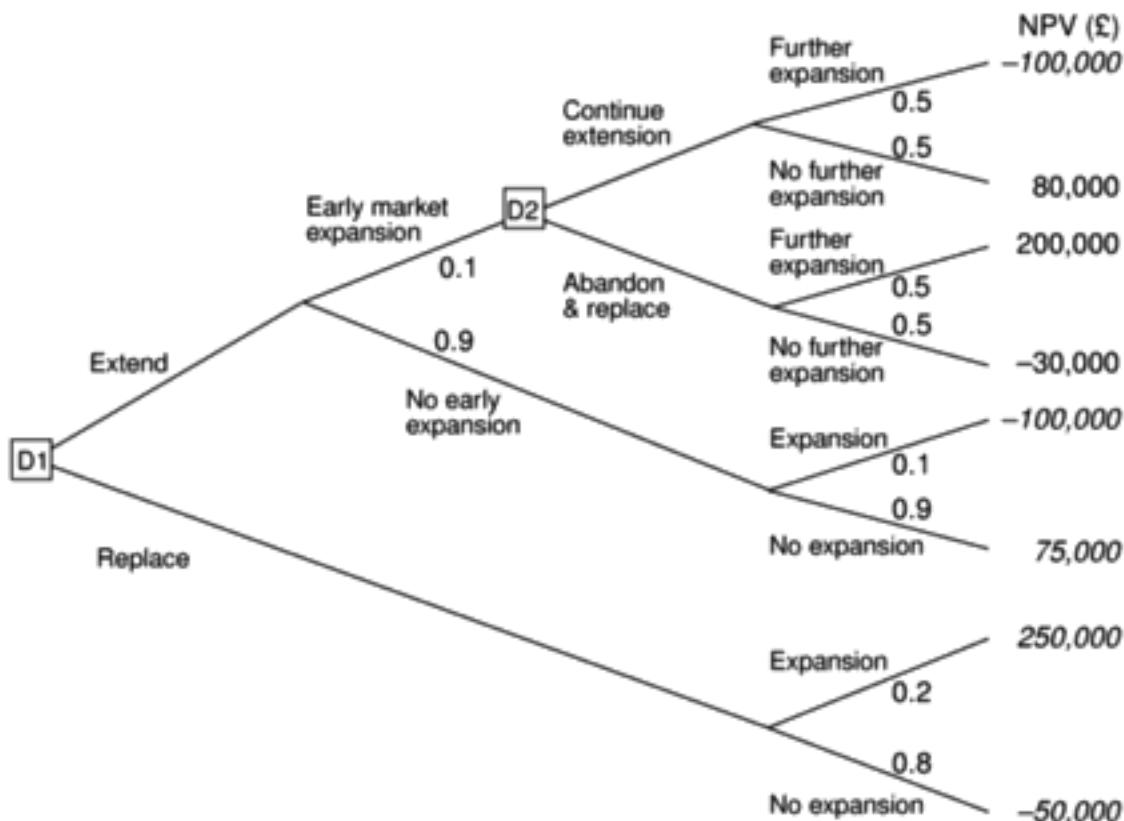


Figure 3.5 A decision tree.

model and analyse problems is the ease with which they can be extended. Figure 3.6 illustrates an extended version of Amanda's decision tree, which includes the possibility of a later decision should they decide to extend the system and then find there is an early market expansion.



The net present values shown in italic are those identified in Amanda's original decision tree shown in Figure 3.5.

Figure 3.6 An extension to Amanda's decision tree.

3.8 Conclusion

Some of the key points in this chapter are:

- projects must be evaluated on strategic, technical and economic grounds;
- economic assessment involves the identification of all costs and income over the lifetime of the system, including its development and operation and checking that the total value of benefits exceeds total expenditure;
- money received in the future is worth less than the same amount of money in hand now, which may be invested to earn interest;
- the uncertainty surrounding estimates of future returns lowers their real value measured now;
- discounted cash flow techniques may be used to evaluate the present value of future cash flows taking account of interest rates and uncertainty;
- cost–benefit analysis techniques and decision trees provide tools for evaluating expected outcomes and choosing between alternative strategies.

3.9 Further exercises

1. Identify the major risks that could affect the success of the Brightmouth College Payroll project and try to rank them in order of importance.
2. Working in a group of three or four, imagine that you are about to embark upon a programming assignment as part of the assessed work for your course. Draw up a list of the risks that might affect the assignment outcome. Individually classify the importance and likelihood of each of those risk as high, medium or low. When you have done this compare your results and try to come up with an agreed project risk matrix.
3. Explain why discounted cash flow techniques provide better criteria for project selection than net profit or return on investment.
4. Consider the decision tree shown in Figure 3.6 and decide, given the additional possibilities, which option(s) IOE should choose.

Chapter 4

Selection of an appropriate project approach

OBJECTIVES

When you have completed this chapter you will be able to:

- take account of the characteristics of the system to be developed when planning a project;
 - select an appropriate process model;
 - make best use of the ‘waterfall’ process model where appropriate;
 - reduce risks by the creation of appropriate prototypes;
 - reduce other risks by implementing the project in increments.
-

4.1 Introduction

The development of software in-house suggests that the project has certain characteristics:

- the project team and the users belong to the same organization;
- the projects being considered slot into a portfolio of existing computer-based systems;
- the methodologies and technologies to be used are not selected by the project manager, but are dictated by local standards.

However, where a series of development projects is being carried out by a software house for different external customers, the methodologies and technologies to be used will have to be decided for each individual project. This decision-making process has been called ‘technical planning’ by some, although here we use the term ‘project analysis’. Even where development is in-house, it is important to spend some time looking for any characteristics of the new project that might make us take a different approach from that used on previous projects. It is this analysis that is the subject of this chapter.

Martyn Ould in *Strategies for Software Engineering: the Management of Risk & Quality*, John Wiley & Sons, 1990, describes how technical planning was done at the software house, Praxis.

The relevant part of the Step Wise approach is Step 3: Analyse project characteristics. The selection of a particular process model will add new products to the Project Breakdown Structure or new activities to the activity network. This will create outputs for Step 4: Identify the products and activities of the project (see Figure 4.1).

In the remainder of this chapter we will firstly look at how the characteristics of a project will influence the approach to the planning of a project. We will then look at some of the most common *process models*, namely the waterfall approach, prototyping and incremental delivery.

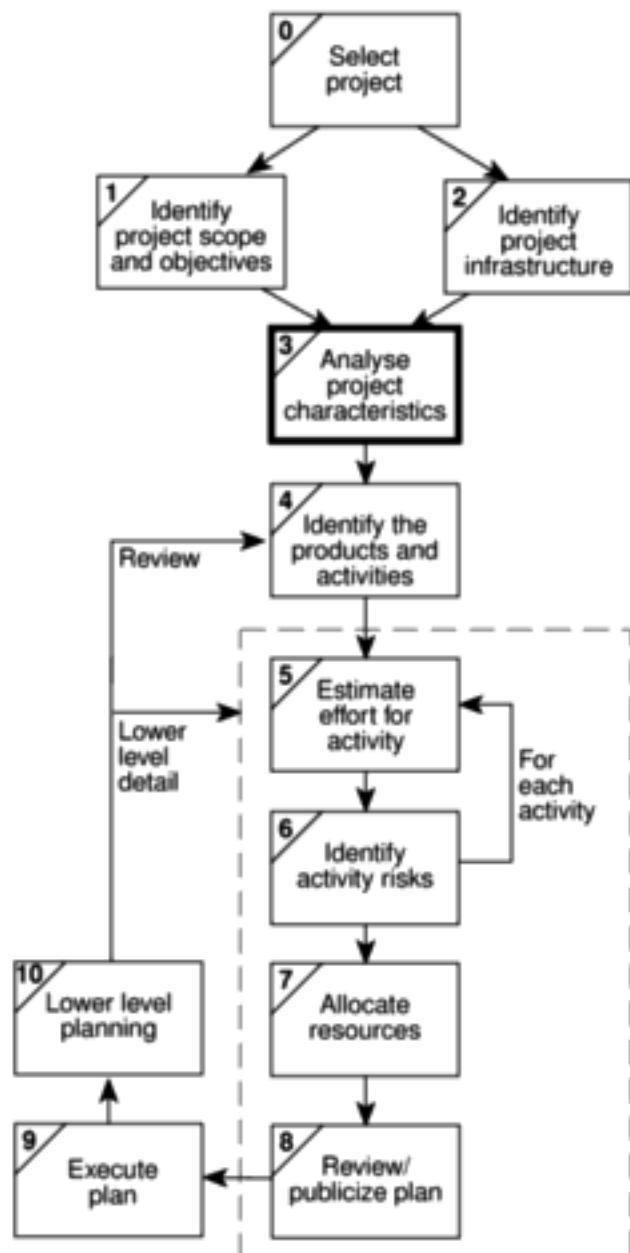


Figure 4.1 Project analysis is the subject of Step 3.

4.2 Choosing technologies

An outcome of project analysis will be the selection of the most appropriate methodologies and technologies. Methodologies include techniques like the various flavours of object-oriented (OO) development, SSADM and JSP (Jackson Structured Programming) while technologies might include an appropriate application-building environment, or the use of knowledge-based system tools.

As well as the products and activities, the chosen technology will influence the following aspects of a project:

- the training requirements for development staff;
- the types of staff to be recruited;
- the development environment – both hardware and software;
- system maintenance arrangements.

We are now going to describe some of the steps of project analysis.

Identify project as either objectives-driven or product-driven

You will recall from Chapter 1 that we distinguished between objectives-driven and product-driven projects. Very often a product-driven project will have been preceded by an objectives-driven project which chose the general software solution that is to be implemented.

There will be cases where things are so vague that even the objectives of the project are uncertain or are the subject of disagreement. People may be experiencing a lot of problems but no-one knows exactly what the solution to the problems might be. It could be that the IT specialists can provide help in some places but assistance from other specialisms is needed in others. In these kinds of situation a *soft systems* approach might need to be considered.

The soft systems approach is described in Checkland, P. and Scholes, J., *Soft systems methodology in action*, John Wiley and Sons, 1990.

Analyse other project characteristics

The sorts of question that would need to be asked include the following.

- **Is a data orientated or a control orientated system to be implemented?** ‘Data orientated’ systems generally mean information systems that will have a considerable database. ‘Control orientated’ systems refer to embedded control systems. These days it is not uncommon to have systems with components of both types.
- **Will the software that is to be produced be a general package or application specific?** An example of a general package would be a spreadsheet or a word processing package. An application specific package could be, for example, an airline seat reservation system.
- **Is the system to be implemented of a particular type for which specific tools have been developed?** For example:

We first introduced the difference between information systems and embedded systems in Chapter 1.

- *does it involve concurrent processing?* – if so the use of techniques appropriate to the analysis and design of such systems would be considered;
- *will the system to be created be knowledge-based?* – expert systems have a set of rules which result in some ‘expert advice’ when applied to a problem domain (sets of methods and tools have been developed to assist in the creation of such systems); or
- will the system to be produced make heavy use of computer graphics?
- **Is the system to be created safety-critical?** For instance, could a malfunction in the system endanger human life?
- **What is the nature of the hardware/software environment in which the system will operate?** It might be that the environment in which the final software will operate is different from that in which it will be developed. Embedded software may be developed on a large development machine that has lots of supporting software tools in the way of compilers, debuggers, static analysers and so on, but might then be down-loaded to a small processor in the larger engineered product. A system destined for a personal computer will need a different approach to one destined for a main-frame or a client-server environment.

Exercise 4.1

How would you categorize each of the following systems according to the classification above?

- (a) a payroll system
 - (b) a system to control a bottling plant
 - (c) a system that holds details of the plans of plant used by a water company to supply water to consumers
 - (d) a software application to support project managers
 - (e) a system used by lawyers to get hold of case law relating to company taxation.
-

Identify high level project risks

Chapter 3 has already touched on some aspects of risk, which are developed further in Chapter 8.

When we first embark on a project we might be expected to work out elaborate plans even though we are at least partially ignorant of many important factors that will affect the project. For example, until we do a detailed investigation of the users’ requirements we will not be able to estimate how much effort will be needed to build a system to meet those requirements. The greater the uncertainties at the beginning of the project, the greater the risk that the project will be unsuccessful. Once we recognize a particular area of uncertainty we can, however, take steps to reduce its uncertainty.

One suggestion is that uncertainty can be associated with the *products*, *processes*, or *resources* associated with the project.

Product uncertainty Here we ask how well the requirements are understood. It might be that the users themselves are uncertain about what a proposed information system is to do. The government, say, might introduce a new form of taxation but the way this is going to operate in detail will not be known until a certain amount of case law has been built up. Some environments can change so quickly that what was a precise and valid statement of requirements rapidly becomes out of date.

Process uncertainty It might be that the project under consideration is the first where an organization has tried to use a method, such as SSADM or OMT, that is new to them. Perhaps a new application building tool is being used. Any change in the way that the systems are developed is going to introduce uncertainty.

OMT is an object-oriented design approach.

Resource uncertainty The main area of uncertainty here will almost surely be the availability of staff of the right ability and experience. A major influence on the degree of uncertainty in a project will be the sheer size of a project. The larger the number of resources needed or the longer the duration of the project, the more inherently risky it is likely to be.

Euromethod, which is reviewed briefly in Appendix C, distinguishes between factors that increase *uncertainty*, for example, continually changing requirements, and those that increase *complexity*, for example, software size. This is because it suggests different strategies to deal with the two distinct types of risk.

At IOE, Amanda has identified possible staff resistance as a risk to the maintenance group accounts project. Would you classify this as a product, process or resource risk? Perhaps it does not fit into any of these categories and some other is needed.

Exercise 4.2

Brigette at Brightmouth College identified the possibility that no suitable payroll package would be available on the market as a risk. What other risks might be inherent in the Brightmouth College payroll project?

Take into account user requirements concerning implementation

A user organization lays down standards that have to be adopted by any contractor providing software for them. For example the UK Civil Service favours the SSADM standard where information systems are being developed.

It is common for organizations to specify that suppliers of software have BS EN ISO 9001:1994 or TickIT accreditation. This will affect the way projects are conducted.

Chapter 12, Software quality, discusses BS EN ISO 9001 and TickIT.

Select general life cycle approach

Control systems A real-time system will have to be implemented using an appropriate methodology, for example, Mascot. Real-time systems that employ concurrent processing will use techniques such as Petri nets.

Information systems Similarly, an information system will need a methodology, such as SSADM or Information Engineering, that matches that type of environment. SSADM will be especially appropriate where the project will employ a large number of development staff whose work will need to be co-ordinated: the method lays down in detail what needs to be done and what products need to be created at each step. Team members would therefore know exactly what is expected of them.

General applications Where the software to be produced is for the general market rather than for a specific application and user, then a methodology such as SSADM would have to be thought about very carefully. This is because the framers of the method make the assumption that a specific user exists. Some parts in the method also assume that there is an existing system that can be analysed to yield the logical features of the new, computer-based, system.

Specialized techniques These have been invented to expedite the development of, for example, *knowledge-based systems* where there are a number of specialized tools and logic based programming languages that can be used to implement this type of system. Similarly a number of specialized techniques and tools have been developed to assist in the development of *graphics-based systems*.

Hardware environment The environment in which the system is to operate can put constraints on the way it is to be implemented. For instance, the need for a fast response time or for the software to take up only a small part of computer memory may mean that only low-level programming languages can be used – particularly in real-time and embedded systems.

Safety-critical systems Where safety and reliability are of the essence, it might be possible to justify the additional expense of a formal specification using a notation such as Z or VDM. Really critical systems call for expensive measures such as having independent teams develop parallel systems with the same functionality. The parallel systems can then run concurrently when the application is in operation so that the results of each of the parallel systems can be cross-checked.

Imprecise requirements Uncertainties or a *novel hardware/software platform* may mean that a *prototyping* approach should be considered. If the environment in which the system is to be implemented is a rapidly changing one, then serious consideration would need to be given to *incremental delivery*. If the users have *uncertain objectives* in connection with the project, then a *soft systems* approach might be desirable.

The implications of prototyping and the incremental approach are explored later in the chapter.

Exercise 4.3

Bearing in mind the discussion above, what, in broad outline, is the most suitable approach for each of the following?

- (a) a system that calculates the amount of a drug that should be administered to a patient who has a particular complaint;

- (b) a system to administer a student loans scheme;
 - (c) a system to control trains on a railway network.
-

4.3 Technical plan contents list

The analysis described above will produce a number of practical requirements that will be fed into the next stage of the planning process. These requirements might add activities to the project and might involve the acquisition of items of software or hardware or the adoption of particular methodologies which might require staff training. As these recommendations imply certain costs, they should be recorded formally.

A preliminary version of this technical plan can be produced by a software house to help in the preparation of the bid for a contract. In some cases, it may actually be shown to the potential customer in order to show the basis for the bid price and generally to impress the customer with the soundness of the approach the software house intends to adopt.

The technical plan is likely to have the following contents.

1. Introduction and summary of constraints:
 - (a) character of the system to be developed;
 - (b) risks and uncertainties of the project;
 - (c) user requirements concerning implementation.
2. Recommended approach:
 - (a) selected methodology or process model;
 - (b) development methods;
 - (c) required software tools;
 - (d) target hardware/software environment.
3. Implementation:
 - (a) required development environment;
 - (b) required maintenance environment;
 - (c) required training.
4. Implications:
 - (a) project products and activities – these will have an effect on the schedule duration and overall project effort;
 - (b) financial – this report will be used to produce costings.

4.4 Choice of process models

The word 'process' is sometimes used to emphasize the idea of a system *in action*. In order to achieve an outcome, the system will have to execute one or more activities: this is its process. This idea can be applied to the development of computer-based systems where a number of interrelated activities have to be

undertaken to create a final product. These activities can be organized in different ways and we can call these *process models*.

A major part of the planning will be the choosing of the development methods to be used and the slotting of these into an overall process model.

The planner needs not only to select methods but also to specify how the method is to be applied. With methods such as SSADM, there is a considerable degree of choice about how it is to be applied: not all parts of SSADM are compulsory. Many student projects have the rather basic failing that at the planning stage they claim that, say, SSADM is to be used: in the event, all that is produced are a few SSADM fragments such as a top level data flow diagram and a preliminary logical data structure diagram. If this is all the particular project requires, it should be stated at the outset.

4.5 Structured methods

Although some 'object-oriented' specialists may object(!), we include the OO approach as a structured method – after all, we hope it is not unstructured. Structured methods are made up of sets of steps and rules, which, when applied produce system products such as data flow diagrams. Each of these products is carefully documented. Such methods are often time consuming compared to more intuitive approaches and this implies some additional cost. The pay-off is such things as a less error prone and more maintainable final system. This balance of costs and benefits is more likely to be justified on a large project involving many developers and users. This is not to say that smaller projects cannot justify the use of such methods.

4.6 Rapid application development

It might be thought that users would generally welcome the more professional approach that structured methods imply. However, customers of IS/IT are concerned with getting working business applications delivered quickly and at less cost and often see structured methods as unnecessarily bureaucratic and slow. A response to this has been *rapid application development* (RAD).

The RAD approach does not preclude the use of some elements of structured methods such as Logical Data Structure diagrams but also adopts tactics such as *joint application development* (JAD) workshops. In these workshops, developers and users work together intensively for, say, three to five days and identify and agree fully documented business requirements. Often, these workshops are conducted away from the normal business and development environments in *clean rooms*, that is, special conference rooms free from outside interruption and suitably furnished with white boards and other aids to communication. This is based on the belief that communication and negotiation that might take several weeks or months via a conventional approach of formal reports and proposals and counter-proposals can be speeded up in these hothouse conditions.

Another practice associated with RAD is *time-boxing* where the scope of a project is rigidly constrained by a pre-agreed and relatively close absolute deadline. Requirements that have to be omitted in order to meet this deadline are considered in later time-boxes.

Some of these ideas will be considered further in our later discussions on prototyping and the incremental approach.

4.7 The waterfall model

This is the ‘classical’ model of system development. An alternative name for this model is the *one-shot* approach. As can be seen from the example in Figure 4.2, there is a sequence of activities working from top to bottom. The diagram shows some arrows pointing upwards and backwards. This indicates that a later stage might reveal the need for some extra work at an earlier stage, but this should definitely be the exception rather than the rule. After all, the flow of a waterfall should be downwards with the possibility of just a little splashing back. The limited scope for iteration is in fact one of the strengths of this process model. With a large project you want to avoid having to go back and rework tasks that you thought had been completed. For a start, having to reopen what was previously thought to be a completed activity plays havoc with promised completion dates.

The first description of this approach is said to be that of H. D. Bennington in ‘Production of Large Computer Programs’ in 1956. This was reprinted in 1983 in *Annals of the History of Computing* 5(4).

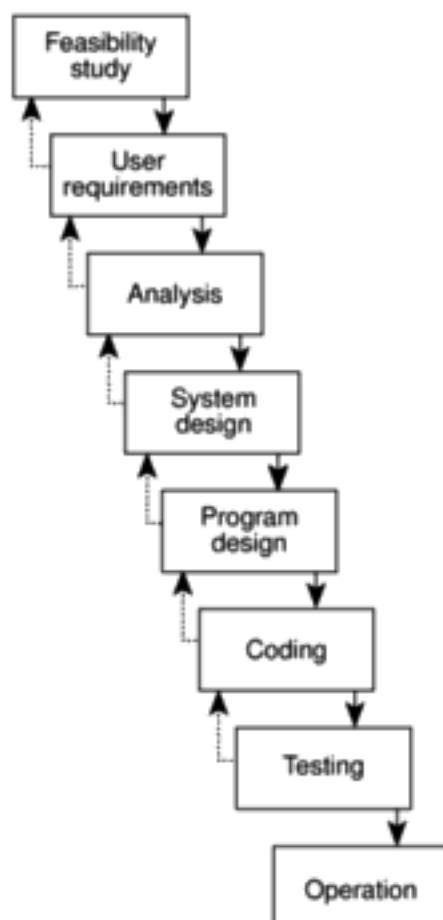


Figure 4.2 The waterfall model.

We contend that there is nothing intrinsically wrong with the waterfall approach, even though more recent writers have suggested different models. It is the ideal that the project manager strives for. The waterfall approach allows project completion times to be forecast with more confidence than is the case with some more iterative approaches and this allows projects to be controlled effectively. However, where there is uncertainty about how a system is to be implemented, and unfortunately there very often is, a more flexible, iterative, approach may be required.

4.8 The V-process model

Figure 4.3 gives a diagrammatic representation of this model. This is an elaboration of the waterfall model and stresses the necessity for validation activities that match the activities that create the products of the project.

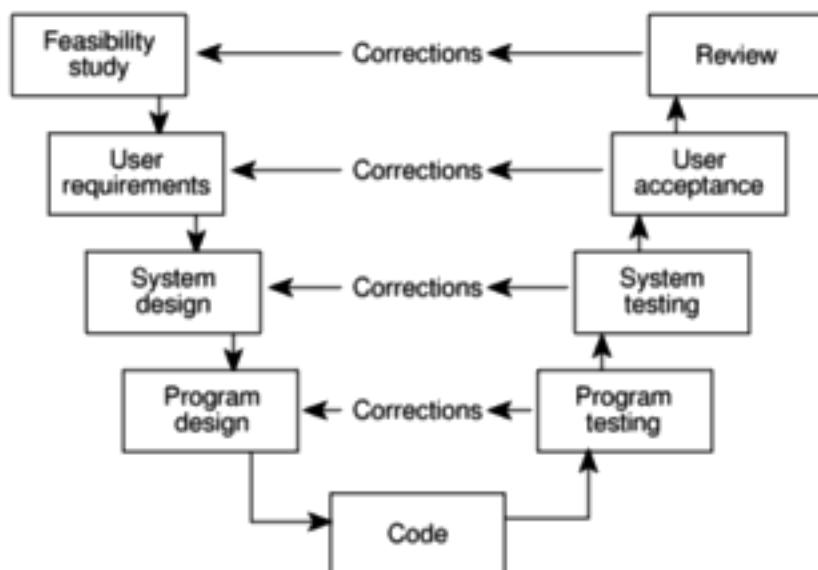


Figure 4.3 *The V-process model.*

The V-process model can be seen as expanding the testing activity in the waterfall model. Each step has a matching validation process that can, where defects are found, cause a loop back to the corresponding development stage and a reworking of the succeeding steps. Ideally, this feeding back should only occur where a discrepancy has been found between what was specified by a particular activity and what was actually implemented in the next lower activity on the descent of the V loop. For example, the system designer might have written that a calculation be carried out in a certain way. The person who structured the software that fulfilled this design might have misunderstood what was required. At system testing stage, the system designer would carry out checks that ensure that the software is doing what was specified in the design document and would discover the program designer's misreading of that document. Only corrections should be

fed back, not the system designer's second thoughts, otherwise the project would be slipping into an 'evolutionary prototyping' approach.

Figure 4.3 shows the V-process model. The review that is held after the system has been implemented is shown as possibly feeding corrections back to the feasibility study which was probably conducted months or years before. How would this work in practice?

Exercise 4.4

4.9 The spiral model

It could be argued that this is another way of looking at the basic waterfall model. In the waterfall model, there is a possible escape at the end of any of the activities in the sequence. A feasibility study might decide that the implementation of a proposed system would be beneficial. The management therefore authorize work on the detailed collection and analysis of user requirements. Some analysis, for instance the interviewing of users, might already have taken place at the feasibility stage, but a more thorough investigation is now launched. This might reveal that in fact the costs of implementing the system would be higher than originally estimated and lead managers to decide to abandon the project.

A greater level of detail is considered at each stage of the project and a greater degree of confidence about the probability of success for the project should be justified. This can be portrayed as a loop or a spiral where the system to be implemented is considered in more and more detail in each sweep and an evaluation process is undertaken before the next iteration is embarked upon. Figure 4.4 illustrates how SSADM can be interpreted in such a way.

The original ideas behind the spiral model can be found in B. W. Boehm's 1988 paper 'A spiral model of software development and enhancement' in *IEEE Computer* 21(5).

4.10 Software prototyping

A prototype is a working model of one or more aspects of the projected system. It is constructed and tested quickly and inexpensively in order to test out assumptions.

Prototypes can be classified as throw-away, evolutionary or incremental.

Throw-away prototypes Here the prototype is used only to test out some ideas and is then discarded when the development of the operational system is commenced. The prototype could be developed using a different software environment (for example, a 4GL as opposed to a 3GL for the final system where machine efficiency is important) or even on a different kind of hardware platform.

Evolutionary prototypes The prototype is developed and modified until it is finally in a state where it can become the operational system. In this case, the standards that are used to develop the software have to be carefully considered.

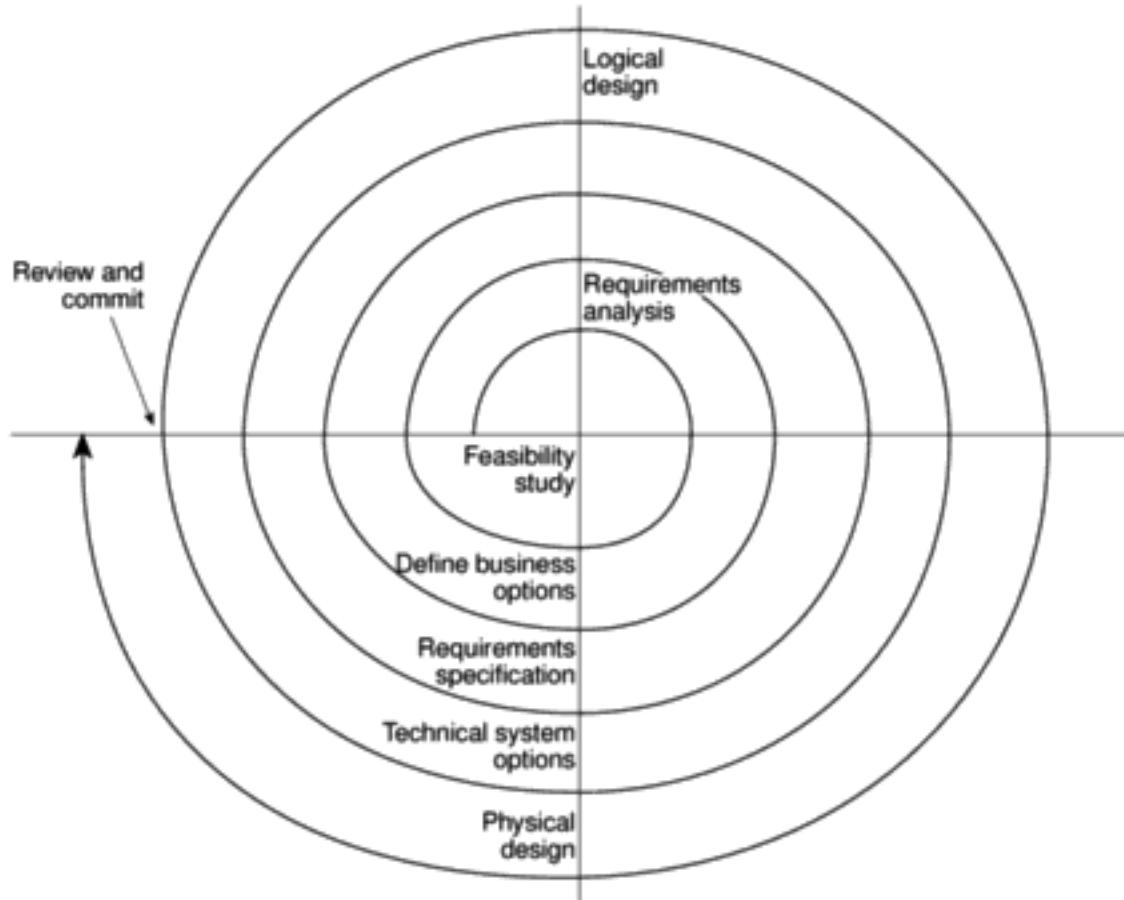


Figure 4.4 The application of the spiral model to SSADM version 4.

Incremental prototypes It could be argued that this is, strictly speaking, not prototyping. The operational system is developed and implemented in small stages so that the feed-back from the earlier stages can influence the development of the later stages.

Some of the reasons that have been put forward for prototyping are the following.

Learning by doing When we have just done something for the first time we can usually look back and see where we have made mistakes.

Improved communication Users are often reluctant to read the massive documents produced by structured methods. Even if they do read this documentation, they do not get a feel for how the system is likely to work in practice.

Improved user involvement The users may be more actively involved in design decisions about the new system.

Clarification of partially-known requirements Where there is no existing system to mimic, users can often get a better idea of what might be useful to them in a potential system by trying out prototypes.

A good book on the general topic of prototyping is R. Vonk's *Prototyping: the effective use of CASE Technology*, Prentice Hall, 1990.

The most important justification for a prototype is the need to reduce uncertainty by conducting an experiment.

Demonstration of the consistency and completeness of a specification Any mechanism that attempts to implement a specification on a computer is likely to uncover ambiguities and omissions.

Reduced need for documentation Because a working prototype can be examined, there is less need for detailed documentation. Some may argue, however, that this is a very dangerous suggestion.

Reduced maintenance costs (that is, changes after the system goes live) If the user is unable to suggest modifications at the prototyping stage the chances are that they will ask for the changes as modifications to the operational system. This reduction of maintenance costs is the main plank in the financial case for creating prototypes.

Feature constraint If an application building tool is used, then the prototype will tend to have features that are easily implemented by that tool. A paper-based design might suggest features that are expensive to implement.

Production of expected results The problem with creating test runs is generally not the creation of the test data but the accurate calculation of the expected results. A prototype can be of assistance here.

Software prototyping is not without its drawbacks and dangers, however.

Users sometimes misunderstand the role of the prototype For example, they might expect the prototype to be as robust as an operational system when incorrect data is entered or they might expect the prototype to have as fast a response as the operational system, although this was not the intention.

Lack of project standards possible Evolutionary prototyping could just be an excuse for a sloppy 'hack it out and see what happens' approach.

Lack of control It is sometimes difficult to control the prototyping cycle as the driving force could be the users' propensity to try out new things.

Additional expense Building and exercising a prototype will incur additional expenses. The additional expense might be less than expected, however, because many analysis and design tasks would have to be undertaken anyway. Some research suggests that typically there is a 10% extra cost to produce a prototype.

Machine efficiency A system built through prototyping, while sensitive to the users' needs, might not be as efficient in machine terms as one developed using more conventional methods.

Close proximity of developers Prototyping often means that code developers have to be sited close to the users. One trend has been for organizations in developed countries to get program coding done cheaply in Third World countries such as India. Prototyping generally will not allow this.

4.11 Other ways of categorizing prototypes

What is being learnt?

The most important reason for having a prototype is that there is a need to learn about an area of uncertainty. For any prototype it is essential that the project managers define at the outset what it is intended to learn from the prototype.

This has a particular relevance to student projects. Students often realize that the software that they are to write as part of their project could not safely be used by real users. They therefore call the software a 'prototype'. However, if it is a real prototype then they must:

- specify what they hope to learn from the prototype;
- plan how the prototype is to be evaluated;
- report on what has actually been learnt.

Prototypes may be used to find out how a new development technique can be used. This would be the case where a new methodology is being used in a pilot scheme. Alternatively, the development methods might be well-known, but the nature of the application might be uncertain.

Different projects will have uncertainties at different stages. Prototypes can therefore be used at different stages. A prototype might be used, for instance, at the requirements gathering stage to pin down requirements that seem blurred and shifting. A prototype might, on the other hand, be used at the design stage to test out the users' ability to navigate through a sequence of input screens.

To what extent is the prototyping to be done?

It would be unusual for the whole of the application to be prototyped. The prototyping might take one of the following forms, which simulates only some aspects of the target application.

Mock-ups For example, copies of the screens that the system is to use are shown to the users on a terminal, but the screens cannot actually be used.

Simulated interaction For example, the user can type in a request to access a record and the system will respond by showing the details of a record, but the details shown are always the same and no access is made to a database.

Partial working model:

- *vertical* – some features are prototyped fully
- *horizontal* – all features are prototyped but not in detail (for example, there might not be a full validation of input).

What is being prototyped?

The human-computer interface With information systems, what the system is to do has usually been established and agreed by management at a fairly early

stage in the development of the system. Prototyping tends to be confined to establishing the precise way in which the operators are to interact with the system. In this case, it is important that the physical vehicle for the prototype be as similar as possible to the operational system.

The functionality of the system In other cases, the precise way that the system should function internally will not be known. This will particularly be the case where a computer model of some real-world phenomenon is being developed. The algorithms used need to be repeatedly adjusted until they satisfactorily imitate the behaviour they should be modelling.

At what stage of a system development project (for example, feasibility study, requirements analysis etc.) would a prototype be useful as a means of reducing the following uncertainties?

Exercise 4.5

- (a) There is a proposal that the senior managers of an insurance company have personal access to management information through an executive information system installed on personal computers located on their desks. Such a system would be costly to set up and there is some doubt about whether the managers would actually use the system.
 - (b) A computer system is to support sales office staff who take phone calls from members of the public enquiring about motor insurance and give quotations over the phone.
 - (c) The insurance company is considering implementing the telephone sales system using the system development features supplied by Microsoft Access. They are not sure, at the moment, that it can provide the kind of interface that would be needed and are also concerned about the possible response times of a system developed using Microsoft Access.
-

4.12 Tools

Special tools are not essential for prototyping but some have made it more practicable.

Application building tools have many features that allow simple computer-based information systems to be set up quickly so that they can be demonstrated to the staff who will use them. An application written in a 3GL becomes more and more complicated as changes accumulate in its code and this makes the software more difficult to alter safely, while applications implemented using application builders that are often form- and table-driven remain flexible.

It has been suggested that the ease of use of some 4GL application builders and the increasing IT awareness of end users might allow end users to create their own prototypes.

Chapter 9 explores configuration management further.

Where a prototype is being constantly tested by the users and modified by the developers there is a need to be able to control the different versions being produced and where necessary to back-track to previous versions.

4.13 A prototyping example

Details of the study can be found in Mayhew, P. J., Worseley, C. J. & Dearnley, P. A. 'Control of Software Prototyping Process: Change Classification Approach'. *Information and Systems Technology* 13(2) 59-66 published in 1989.

The use of prototyping on the COMET Commercial Estimating System project at the Royal Dockyards was the subject of a study, information about which has been published. The Royal Dockyards had been transferred to a system of 'commercial management' where they had to produce estimates of the cost of doing tasks that could then be compared against the cost of contracting the work out. Staff at the dockyards were inexperienced with this method of working.

It was decided to implement a computer system to support the estimating process and it was realized that the human-computer interface would be important. For this reason, the software house implementing the system suggested a prototyping approach.

Among the questions that had to be decided was who should be present at evaluation sessions. If managers were there, would they dominate the proceedings at the expense of those who would actually use the system? Furthermore, if the designers were present they might well feel defensive about the system presented and try and argue against changes suggested.

The results of the prototype

The impact of prototyping may be judged by the fact that although the preliminary design had been done using SSADM, as a result of the evaluation of the prototype the number of screens in the system was doubled.

A major problem was that of controlling changes to the prototype. In order to record and control the changes suggested by users, the changes were categorized into three types.

Cosmetic (about 35% of changes). These were simply changes to the layout of the screen. They were:

- implemented;
- recorded.

Local (about 60% of changes). These involved changes to the way that the screen was processed but did not affect other parts of the system. They were:

- implemented;
- recorded;
- backed-up so that they could be removed at a later stage if necessary;
- inspected retrospectively.

Global (about 5% of changes). These were changes that affected more than one part of the processing. All changes here had to be the subject of a design review before they could be implemented.

4.14 Incremental delivery

This is similar to the 'incremental prototyping' approach mentioned above. One of the most prominent advocates of this approach is Tom Gilb. The approach involves breaking the system down into small components which are then implemented and delivered in sequence. Each component that is delivered must actually give some benefit to the user. Figure 4.5 gives a general idea of the approach.

Principles of Software Engineering Management by Tom Gilb, published by Addison-Wesley in 1988, argues strongly in favour of this approach.

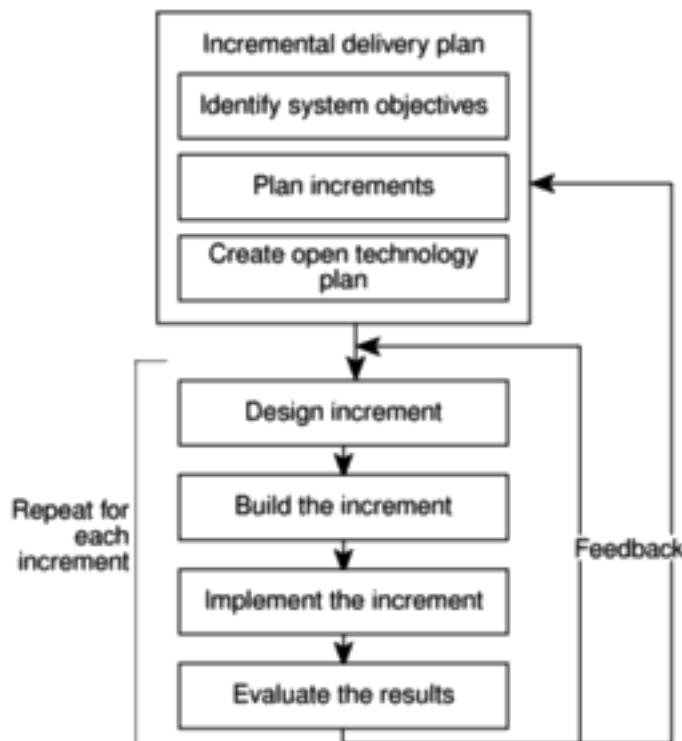


Figure 4.5 Intentional incremental delivery.

Advantages of this approach

These are some of the justifications given for the approach:

- the feedback from early increments can influence the later stages;
- the possibility of changes in requirements is not so great as with large monolithic projects because of the shorter timespan between the design of a component and its delivery;
- users get benefits earlier than with a conventional approach;
- early delivery of some useful components improves cash flow, because you get some return on investment early on;

- smaller sub-projects are easier to control and manage;
- 'gold-plating', the requesting of features that are unnecessary and not in fact used, should be less as users will know that they get more than one opportunity to make their requirements known: if a feature is not in the current increment then it can be included in the next;
- the project can be temporarily abandoned if more urgent work crops up;
- job satisfaction is increased for developers who see their labours bearing fruit at regular, short, intervals.

Disadvantages

On the other hand these disadvantages have been put forward:

- 'software breakage', that is, later increments might require the earlier increments to be modified;
- developers might be more productive working on one large system than on a series of smaller ones.

The incremental delivery plan

The content of each increment and the order in which the increments are to be delivered to the users of the system have to be planned at the outset.

Basically the same process has to be undertaken as in strategic planning but at a more detailed level where the attention is given to increments of a user application rather than whole applications. The elements of the incremental plan are the *system objectives*, *incremental plan* and the *open technology plan*.

Identify system objectives

The purpose is to give an idea of the 'big picture', that is, the overall objectives that the system is to achieve. These can then be expanded into more specific functional goals and quality goals.

Functional goals will include:

- objectives it is intended to achieve;
- jobs the system is to do;
- computer/non-computer functions to achieve them.

In addition, measurable quality characteristics should be defined, such as reliability, response and security. This reflects Tom Gilb's concern that system developers always keep sight of the objectives that they are trying to achieve on behalf of their clients. In the quickly changing environment of an application, individual requirements may change over the course of the project, but the objectives should not.

Plan increments

Having defined the overall objectives, the next stage is to plan the increments using the following guidelines:

- steps typically should consist of 1% to 5% of the total project;
- non-computer steps may be included – but these must deliver benefits directly to the users;
- ideally, an increment should take one month or less and should never take more than three months;
- each increment should deliver some benefit to the user;
- some increments will be physically dependent on others;
- value-to-cost ratios may be used to decide priorities (see below).

Very often a new system will be replacing an old computer system and the first increments may use parts of the old system. For example, the data for the database of the new system may initially be obtained from the old system's standing files.

Which steps should be first? Some steps might be prerequisites because of physical dependencies but others can be in any order. Value-to-cost ratios can be used to establish the order in which increments are to be developed. The customer is asked to rate each increment with a score in the range 1–10 in terms of its value. The developers also rate the cost of developing each of the increments with a score in the range 0–10. This might seem a rather crude way of evaluating costs and benefits, but people are often unwilling to be more precise. By then dividing the value rating by the cost rating, a rating which indicates the relative 'value for money' of each increment is derived.

The value to cost ratio = V/C where V is a score 1–10 representing value to customer and C is a score 0–10 representing cost.

Table 4.1 Ranking by value to cost ratio

Step	Value	Cost	Ratio	Rank
Profit reports	9	1	9	(2nd)
Online database	1	9	0.11	(6th)
Ad hoc enquiry	5	5	1	(4th)
Production sequence plans	2	8	0.25	(5th)
Purchasing profit factors	9	4	2.25	(3rd)
Clerical procedures	0	7	0	(7th)
Profit based pay for managers	9	0	∞	(1st)

Create open technology plan

If the system is to be able to cope with new components being continually added then it has to be built so that it is extendible, portable and maintainable.

As a minimum this will require the use of:

- a standard high level language;
- a standard operating system;
- small modules;
- variable parameters, for example, items such as organization name, department names and charge rates are held in a parameter file that can be amended without programmer intervention;
- a standard database management system.

These are all things that might be expected as a matter of course in a modern IS development environment.

4.15 An incremental example

Tom Gilb describes a project where a software house negotiated a fixed price contract with a three-month delivery time with the Swedish Government to supply a system to support map-making. It later became apparent that the original estimate of effort upon which the bid was based was probably about half what the real effort would be.

The project was replanned so that it was divided into ten increments, each supplying something of use to the customer. The final increments were not available until three months after the contract's delivery date. The customer was not in fact unhappy about this as the most important parts of the system had actually been delivered early.

4.16 Selecting the most appropriate process model

See Appendixes C and D for overviews of Euromethod and ISO 12207, both of which offer advice on process model selection.

Both Euromethod and ISO 12207 provide advice on this.

Euromethod distinguishes between the *construction* of an application and its *installation*. It is possible to use different approaches for these two stages. For example, an application could be constructed using a one-shot strategy but then be released to its users in increments. The only combinations of construction and installation strategies that are not feasible are evolutionary installation with any other construction approach than evolutionary.

In general, Euromethod suggests that where uncertainty is high then an evolutionary approach is to be favoured. An example of uncertainty would be where the users' requirements are not clearly defined. Where the requirements are relatively certain but there are many complexities, for example, where there is a large embedded system that is going to need a large amount of code, then an incremental approach might be favoured. Where deadlines are tight, then either an evolutionary or an incremental approach is favoured over a one-shot strategy, as both tactics should allow at least something to be delivered at the deadline, even

if it is not all that was originally promised. Students about to plan final year projects would do well to note this.

4.17 Conclusion

This chapter has stressed the need to examine each project carefully to see if it has characteristics that suggest a particular approach or process model. These characteristics will also suggest the addition of specific activities to the project plan.

The classic waterfall process model, which attempts to minimize iteration, should lead to projects that are easy to control. Unfortunately many projects do not lend themselves to this structure. Prototyping often reduces project uncertainties by allowing knowledge to be bought through experimentation. The incremental approach encourages the execution of a series of small, manageable, 'mini-projects' but does have some costs.

4.18 Further exercises

1. A bank which provides loans to home buyers has a long history of implementing computer-based information systems to support the work of its branches. It uses a proprietary structured systems analysis and design method. It has been decided to create a computer model of the property market. This would attempt for example to calculate the effect of changes of interest rates on house values. There is some concern that the usual methodology used for IS development would not be appropriate for the new project.
 - (a) Why might there be this concern and what other approaches should be considered?
 - (b) Outline a plan for the development of the system that illustrates the application of your preferred methodology for this project.
2. A software application is to be designed and built to assist in software cost estimation. It responds to certain input parameters and produces initial cost estimates to be used at bidding time.
 - (a) It has been suggested that a software prototype would be of value in these circumstances. Explain why this might be.
 - (b) Discuss how such prototyping could be controlled to ensure that it is conducted in an orderly and effective way and within a specified time span.
3. An invoicing system is to have the following components: amend invoice, produce invoice, produce monthly statements, record cash payment, clear paid invoices from database, create customer records, delete customer.
 - (a) What physical dependencies govern the order in which these transactions are implemented?

- (b) How could the system be broken down into increments which would be of some value to the users. Hint – think about the problems of taking existing details onto a database when a system is first implemented.
- 4. What are the features of the following that contribute to an open systems architecture as recommended by Tom Gilb:
 - (a) the UNIX™ operating system;
 - (b) SQL;
 - (c) C++;
 - (d) Jackson Structured Programming.

Chapter 5

Software effort estimation

OBJECTIVES

When you have completed this chapter you will be able to:

- avoid the dangers of unrealistic estimates;
 - understand the range of estimating methods that can be used;
 - estimate projects using a bottom-up approach;
 - count the function points and object points for a system;
 - estimate the effort needed to implement software using a procedural programming language;
 - understand the COCOMO approach to developing effort models.
-

5.1 Introduction

One definition of a successful project is that the system is delivered ‘on time and within budget and with the required quality’, which implies that targets are set and the project leader then tries to meet those targets. This assumes that the targets are reasonable – the possibility of a project leaders achieving record levels of productivity from the team, but still not meeting a deadline because of incorrect initial estimates is not recognized. Realistic estimates are therefore crucial to the project leader.

What sorts of problem might a project leader such as Amanda, who is in charge of the IOE Maintenance Group Accounts project, encounter when trying to do estimates? Estimating the effort required to implement software is notoriously difficult. Some of the difficulties of estimating are inherent in the very nature of software, especially its complexity and invisibility. In addition the intensely human activities that make up system development cannot be treated in a purely mechanistic way. Other difficulties include:

In Chapter 1, the special characteristics of software identified by Brooks, i.e. complexity, conformity, changeability and invisibility, were discussed.

- **Novel applications of software** With traditional engineering projects, it is often the case that the system to be created is similar to one constructed previously but for a different customer or in a different location. The estimates for such a project can therefore be based on previous experience. With software, in most major projects the product will in some way be unique and will therefore be clouded with doubts and uncertainties.
- **Changing technology** For example, at IOE the original maintenance billing system might have been written in Cobol, while the new extension for group accounts that Amanda is directing might be developed using an application building environment such as that provided by Oracle.
- **Lack of homogeneity of project experience** As we will see, effective estimating should be based on information about how past projects have performed. It is surprising how many organizations do not make this data available to staff. Amanda might also find that even where the previous project data is available, it might not be that useful.

Table 5.1 is a set of figures recorded for actual projects carried out by the same organization.

Table 5.1 *Some project data – effort in work months (as percentage of total effort in brackets)*

<i>Project</i>	<i>Design</i>		<i>Coding</i>		<i>Testing</i>		<i>Total</i>	
	<i>wm</i>	(%)	<i>wm</i>	(%)	<i>wm</i>	(%)	<i>wm</i>	<i>SLOC</i>
a	3.9	(23)	5.3	(32)	7.4	(44)	16.7	6050
b	2.7	(12)	13.4	(59)	6.5	(26)	22.6	8363
c	3.5	(11)	26.8	(83)	1.9	(6)	32.2	13334
d	0.8	(21)	2.4	(62)	0.7	(18)	3.9	5942
e	1.8	(10)	7.7	(44)	7.8	(45)	17.3	3315
f	19.0	(28)	29.7	(44)	19.0	(28)	67.7	38988
g	2.1	(21)	7.4	(74)	0.5	(5)	10.1	38614
h	1.3	(7)	12.7	(66)	5.3	(27)	19.3	12762
i	8.5	(14)	22.7	(38)	28.2	(47)	59.5	26500

The figures are taken from
B. A. Kitchenham and N.
R. Taylor 'Software Project
Development Cost
Estimation' in *Journal of
Systems and Software*
1985 (5).
The abbreviation SLOC
stands for 'source lines of
code'. SLOC is one way of
indicating the size of a
system.

Exercise 5.1

Calculate the productivity (that is, SLOC per work month) of each of the projects in Table 5.1 and also for the organization as a whole. If the project leaders for projects a and d had correctly estimated the source number of lines of code (SLOC) and then used the average productivity of the organization to calculate the effort needed to complete the projects, how far out would their estimates have been from the actual effort?

It would be very difficult on the basis of this information to advise a project manager about what sort of productivity to expect, or about the probable distribution of effort among the phases of design, coding and testing that could be expected from a new project.

There have been some attempts to set up industry-wide databases of past projects. However, this data seems to be of limited use to estimators as there are uncertainties in the way that various terms can be interpreted. For example, what exactly is meant by the term 'testing'? Does it cover the activity of the software developer when debugging code? Does 'design' include drawing up program structure diagrams or does this come under the heading of 'programming'?

Subjective nature of estimating Some research shows that people tend to under-estimate the difficulty of small tasks and over-estimate that of large ones. In the world of software development this is perhaps justifiable, as large projects are usually disproportionately more complex and more difficult than smaller ones.

Political implications Different groups within an organization have different objectives. The IOE information systems development managers might, for example, want to see as many systems as possible implemented and will therefore put pressure on estimators to reduce cost estimates. As Amanda is responsible for the development of the maintenance group accounts subsystem, she might be concerned that the project does not exceed its budget and is not delivered late, because this will reflect badly on herself. She might therefore try to increase the estimates. One suggestion is that all estimates should be carried out within an organization by a specialist estimating group, independent of both the users and the project team. Not all agree with this, as staff involved in a project are more likely to be committed to targets where they have participated in formulating them.

The possibility of the different groups with stakes in a project having different and possibly conflicting objectives was discussed in Chapter 1.

5.2 Where are estimates done?

Estimates are carried out at various stages of a software project. At each stage, the reasons for the estimate and the methods used will vary.

Strategic planning The costs of computerizing potential applications as well as the benefits of doing so might need to be estimated to help decide what priority to give to each project. Such estimates might also influence the numbers of various types of development staff to be recruited by the organization.

Chapter 3 discusses strategic planning in some detail.

Feasibility study This ascertains that the benefits of the potential system will justify the costs.

System specification Most system development methodologies usefully distinguish between the definition of the users' requirements and the design that documents how those requirements are to be fulfilled. The effort needed to implement different design proposals will need to be estimated. Estimates at the design stage will also confirm that the feasibility study is still valid, taking into account all that has been learnt during detailed requirements analysis.

The estimate at this stage cannot be based only on the user requirement: some kind of technical plan is also needed – see Chapter 4.

Evaluation of suppliers' proposals In the case of the IOE maintenance group accounts subsystem, for example, IOE might consider putting the actual system-building out to tender. Staff in the software houses that are considering a bid would need to scrutinize the system specification and produce estimates on which to base proposals. Amanda might still be required to carry out her own estimate to help judge the bids received. IOE might wish to question a proposal that seems too low: they might wonder, for example, whether the proposer had properly understood the requirements. If, on the other hand, the bids seem too high, they might reconsider in-house development.

Project planning As the planning and implementation of the project progresses to greater levels of detail, more detailed estimates of smaller work components will be made. As well as confirming the earlier and more broad-brush estimates, these will help answer questions about, for example, when staff will have completed particular tasks and be available for new activities. Two general points can be made here:

- as the project proceeds, so the accuracy of the estimates should improve as knowledge about the nature of the project increases;
- conventional wisdom is that at the beginning of the project the user requirement (that is, a logical model of the required system) is of paramount importance and that premature consideration of the physical implementation is to be avoided. In order to do an estimate, however, the estimator will have to speculate about this physical implementation, for instance about the number of software modules to be written.

To set estimating into the context of the Step Wise framework (Figure 5.1) presented in Chapter 1, re-estimating might take place at almost any step, but specific provision is made for it at Step 3, 'Analyse project characteristics', where a relatively high-level estimate will be produced, and in Step 5 for each individual activity. As Steps 5–8 are repeated at progressively lower levels, so estimates will be done at a finer degree of detail. As we will see later in this chapter, different methods of estimating are needed at these different planning steps.

5.3 Problems with over- and under-estimates

A project leader such as Amanda will need to be aware that the estimate itself, if known to the development team, will influence the time required to implement the system. An over-estimate might cause the project to take longer than it would otherwise. This can be explained by the application of two 'laws'.

Parkinson's Law 'Work expands to fill the time available', which implies that given an easy target staff will work less hard.

Brooks' Law The effort required to implement a project will go up disproportionately with the number of staff assigned to the project. As the project team grows in size so will the effort that has to go into management, co-ordination and communication. This has given rise, in extreme cases, to the notion of Brooks'

Parkinson's law was originally expounded in C. Northcote Parkinson's tongue-in-cheek book *Parkinson's Law*, John Murray, 1957.

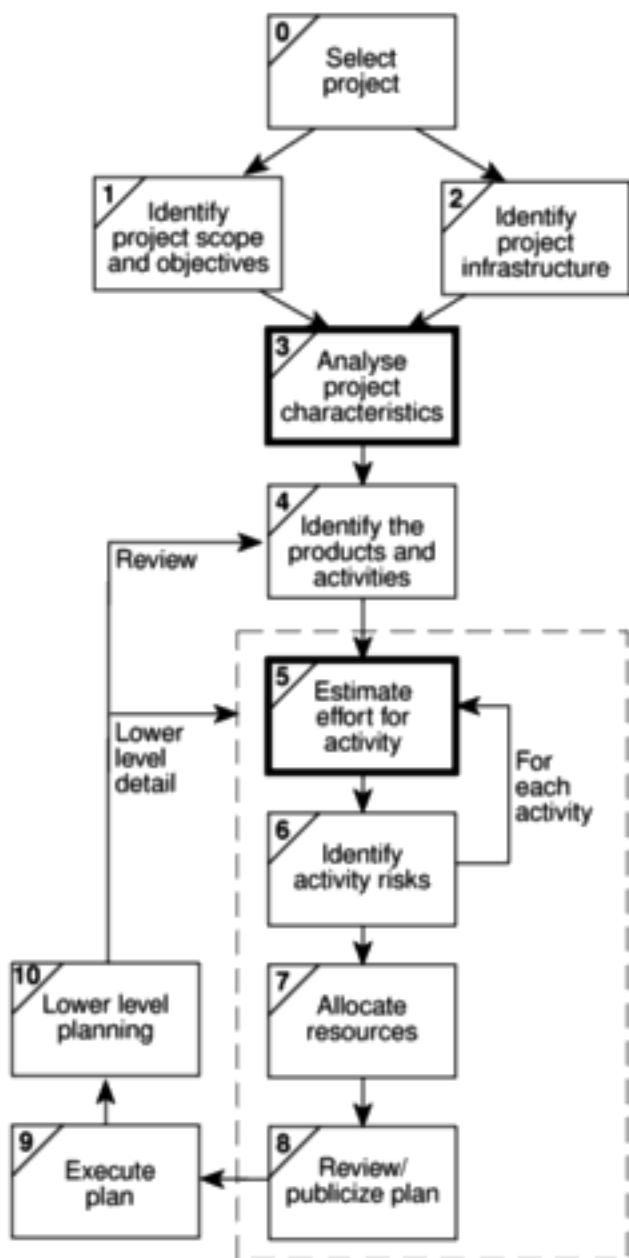


Figure 5.1 Software estimation takes place in Steps 3 and 5 in particular.

Law: 'putting more people on a late job makes it later'. If there is an over-estimate of the effort required then this might lead to more staff being allocated than are needed and managerial overheads will be increased. This is more likely to be of significance with large projects.

Some have suggested that while the under-estimated project might not be completed on time or to cost, it might still be implemented in a shorter time than a project with a more generous estimate. There must, however, be limits to this phenomenon where all the slack in the project is taken up.

The danger with the under-estimate is the effect on quality. Staff, particularly those with less experience, might respond to pressing deadlines by producing work which is sub-standard. Since we are into laws, this might be seen as a manifestation of Weinberg's zeroth law of reliability: 'if a system does not have to

Brooks' law comes from *The Mythical Man-month* that has been referred to already.

See T. K. Hamid and S. E. Madnick 'Lessons learnt from modeling the dynamics of software development' in C. F. Kemerer (ed.) *Software Project Management*, Irwin, 1997.

be reliable, it can meet any other objective'. In other words, if there is no need for the program actually to work, you can meet any programming deadline that might be set! Sub-standard work might only become visible at the later, testing, phases of a project, which are particularly difficult to control and where extensive rework can have catastrophic consequences for the project completion date.

Because of the possible effects on the behaviour of development staff caused by the size of estimates, they might be artificially reduced by their managers to increase pressure on staff. This will work only where staff are unaware that this has been done. Research has found that motivation and morale are enhanced where targets are achievable. If, over a period of time, staff become aware that the targets set are unattainable and that projects are routinely not meeting their published targets, then this will help to destroy motivation. Furthermore, people like to think of themselves as winners and there is a general tendency to put success down to our own efforts, while failure is blamed on the organization.

In the end, an estimate is not really a prediction, it is a *management goal*. Barry Boehm has suggested that if a software development cost is within 20% of the estimated cost estimate for the job then a good manager can turn it into a self-fulfilling prophecy. A project leader like Amanda will work hard to make the actual performance conform to the estimate.

Barry Boehm devised the COCOMO estimating models, which are described later in this chapter.

5.4 The basis for software estimating

The need for historical data

Nearly all estimating methods need information about how projects have been implemented in the past. However, care needs to be taken in judging the applicability of data to the estimator's own circumstances because of possible differences in environmental factors such as the programming languages used, the software tools available, the standards enforced and the experience of the staff.

Measure of work

It is normally not possible to calculate directly the actual cost or time required to implement a project. The time taken to write a program might vary according to the competence or experience of the programmer. Implementation times might also vary because of environmental factors such as the software tools available. The usual practice is therefore to express the work content of the system to be implemented independently of effort, using a measure such as source lines of code (SLOC). The reader might also come across the abbreviation KLOC which refers to thousands of lines of code.

As can be imagined, SLOC is a very imprecise measure. Does it include comment lines? Are data declarations to be included? Unfortunately, researchers have not been consistent on these points. The writers' view is that comment lines are excluded, but data declarations are included. The argument for including data declarations is that in Cobol especially, it is possible to transfer much of the specification of processing to the DATA DIVISION, by using, for example, the

SLOC has already been used in Table 5.1.

R. E. Park has devised a standard for counting source statements that has been widely adopted—see *Software size measurement: a framework for counting source statements*, Software Engineering Institute, 1992.

report writer facility. Others might argue over this, but the main point is that consistency is essential.

Counting SLOC is difficult when employing application building tools, which often use tables or diagrams to record processing rules. Different measures of size are needed, such as object or function points, which are explained further on in this chapter.

Complexity

Two programs with the same KLOC will not necessarily take the same time to write, even if done by the same developer in the same environment. One program might be more complex. Because of this, SLOC estimates have to be modified to take complexity into account. Attempts have been made to find objective measures of complexity, but often it will depend on the subjective judgement of the estimator.

Differences in complexity might be one of the reasons for the inconsistencies in Table 5.1.

5.5 Software effort estimation techniques

Barry Boehm, in his classic work on software effort models, identified the main ways of deriving estimates of software development effort as:

- **algorithmic models** – which use ‘effort drivers’ representing characteristics of the target system and the implementation environment to predict effort;
- **expert judgement** – where the advice of knowledgeable staff is solicited;
- **analogy** – where a similar, completed, project is identified and its actual effort is used as a basis for the new project;
- **Parkinson** – which identifies the staff effort available to do a project and uses that as the ‘estimate’;
- **price to win** – where the ‘estimate’ is a figure that appears to be sufficiently low to win a contract;
- **top-down** – where an overall estimate is formulated for the whole project and is then broken down into the effort required for component tasks;
- **bottom-up** – where component tasks are identified and sized and these individual estimates are aggregated.

See B. W. Boehm
‘Software Engineering Economics’ in C. F. Kemerer (ed.) *Software Project Management*, Irwin, 1997.

Clearly, the ‘Parkinson’ method is not really an effort prediction method, but a method of setting the scope of a project. Similarly, ‘price to win’ is a way of deciding a price and not a prediction method. On these grounds, Boehm rejects them as prediction techniques although they might have some value as management techniques. There is, for example, a perfectly acceptable engineering practice of ‘design to cost’ which is one example of the broader approach of ‘design by objectives’.

We will now look at some of these techniques more closely. First we will examine the difference between top-down and bottom-up estimating.

Bottom-up estimating

Estimating methods can be generally divided into bottom-up and top-down approaches. With the bottom-up approach, the estimator breaks the project into its component tasks and then estimates how much effort will be required to carry out each task. With a large project, the process of breaking down into tasks would be a repetitive one: each task would be analysed into its component sub-tasks and these in turn would be further analysed. This is repeated until you get to components that can be executed by a single person in about a week or two. The reader might wonder why this is not called a top-down approach: after all you are starting from the top and working down! Although this top-down analysis is an essential precursor to bottom-up estimating, it is really a separate one – that of producing a Work Breakdown Structure (WBS). The bottom-up part comes in adding up the calculated effort for each activity to get an overall estimate.

The bottom-up approach is most appropriate at the later, more detailed, stages of project planning. If this method is used early on in the project cycle then the estimator will have to make some assumptions about the characteristics of the final system, for example the number and size of software modules. These will be working assumptions that imply no commitment when it comes to the actual design of the system.

Where a project is completely novel or there is no historical data available, the estimator would be well advised to use the bottom-up approach.

Exercise 5.2

Brigette at Brightmouth College has been told that there is a requirement, now that the payroll system has been successfully installed, to create a subsystem that analyses the staffing costs for each course. Details of the pay that each member of staff receives can be obtained from the payroll standing data. The number of hours that each member of staff spends teaching on each course can be obtained from standing files in a computer-based time-tabling system.

What tasks would have to be undertaken to implement this requirement? Try to identify tasks that would take one person about 1 or 2 weeks.

Which tasks are the ones whose durations are most difficult to estimate?

The top-down approach and parametric models

The top-down approach is normally associated with parametric (or algorithmic) models. These may be explained using the analogy of estimating the cost of rebuilding a house. This would be of practical concern to a house-owner who needs sufficient insurance cover to allow for rebuilding the property if it were destroyed. Unless the house-owner happens to be in the building trade it is unlikely that he or she would be able to work out how many bricklayer-hours, how many carpenter-hours, electrician-hours and so on would be required. Insurance companies, however, produce convenient tables where the house-owner can find

an estimate of rebuilding costs based on such *parameters* as the number of storeys and the floor space that a house has. This is a simple parametric model.

The effort needed to implement a project will be related mainly to variables associated with characteristics of the final system. The form of the parametric model will normally be one or more formulae in the form:

$$\text{effort} = (\text{system size}) \times (\text{productivity rate})$$

For example, system size might be in the form 'thousands of lines of code' (KLOC) and the productivity rate 40 days per KLOC. The values to be used will often be matters of subjective judgement.

A model to forecast software development effort therefore has two key components. The first is a method of assessing the size of the software development task to be undertaken. The second assesses the rate of work at which the task can be done. For example, Amanda at IOE might estimate that the first software module to be constructed is 2 KLOC. She might then judge that if Kate undertook the development of the code, with her expertise she could work at a rate of 40 days per KLOC and complete the work in 2×40 days, that is, 80 days, while Ken, who is less experienced, would need 55 days per KLOC and take 2×55 that is, 110 days to complete the task.

Some parametric models, such as that implied by function points, are focused on system or task size, while others, such as COCOMO, are more concerned with productivity factors.

Having calculated the overall effort required, the problem is then to allocate proportions of that effort to the various activities within that project.

The top-down and bottom-up approaches are not mutually exclusive. Project managers will probably try to get a number of different estimates from different people using different methods. Some parts of an overall estimate could be derived using a top-down approach while other parts could be calculated using a bottom-up method.

At the earlier stages of a project, the top-down approach would tend to be used, while at later stages the bottom-up approach might be preferred.

Students on a course are required to produce a written report on an IT-related topic each semester. If you wanted to create a model to estimate how long it should take a student to complete such an assignment, what measure of work content would you use? Some reports might be more difficult to produce than others: what factors might affect the degree of difficulty?

Exercise 5.3

5.6 Expert judgement

This is asking someone who is knowledgeable about either the application area or the development environment to give an estimate of the effort needed to carry out a task. This method will most likely be used when estimating the effort needed to

change an existing piece of software. The estimator would have to carry out some kind of impact analysis in order to judge the proportion of code that would be affected and from that derive an estimate. Someone already familiar with the software would be in the best position to do this.

Some have suggested that expert judgement is simply a matter of guessing, but our own research has shown that experts tend to use a combination of an informal analogy approach where similar projects from the past are identified (see below), and bottom-up estimating.

5.7 Estimating by analogy

The use of analogy is also called *case-based reasoning*. The estimator seeks out projects that have been completed (*source cases*) and that have similar characteristics to the new project (the *target case*). The effort that has been recorded for the matching source case can then be used as a base estimate for the target. The estimator should then try to identify any differences between the target and the source and make adjustments to the base estimate for the new project.

This might be a good approach where you have information about some previous projects but not enough to draw generalized conclusions about what variables might make good size parameters.

A problem here is how you actually identify the similarities and differences between the different systems. Attempts have been made to automate this process. One software application that has been developed to do this is ANGEL. This identifies the source case that is nearest the target by measuring the Euclidean distance between cases. The source case that is at the shortest Euclidean distance from the target is deemed to be the closest match. The Euclidean distance is calculated:

$$\text{distance} = \text{square-root of } ((\text{target_parameter}_1 - \text{source_parameter}_1)^2 + \dots + (\text{target_parameter}_n - \text{source_parameter}_n)^2)$$

See M. Shepperd and C. Schofield 'Estimating software project effort using analogies' in *IEEE Transactions in Software Engineering*, 1997 SE-23(11) pp 736–743

Example 5.1

Say that the cases are being matched on the basis of two parameters, the number of inputs to and the number of outputs from the system to be built. The new project is known to require 7 inputs and 15 outputs. One of the past cases, Project A, has 8 inputs and 17 outputs. The Euclidean distance between the source and the target is therefore the square-root of $((7-8)^2 + (17-15)^2)$, that is 2.24.

Exercise 5.4

Project B has 5 inputs and 10 outputs. What would be the Euclidean distance between this project and the target new project being considered above? Is Project B a better analogy with the target than Project A?

The above explanation is simply to give an idea of how Euclidean distance might be calculated. The ANGEL package uses rather more sophisticated algorithms based on this principle.

5.8 Albrecht function point analysis

This is a top-down method that was devised by Allan Albrecht when he worked for IBM. Albrecht was investigating programming productivity and needed some way to quantify the functional size of programs independently of the programming languages in which they had been coded. He developed the idea of *function points* (FPs).

The basis of function point analysis is that computer-based information systems comprise five major components, or *external user types* in Albrecht's terminology, that are of benefit to the users:

- **External input types** are input transactions that update internal computer files.
- **External output types** are transactions where data is output to the user. Typically these would be printed reports, since screen displays would come under external inquiry types (see below).
- **Logical internal file types** are the standing files used by the system. The term 'file' does not sit easily with modern information systems. It refers to a group of data that is usually accessed together. It might be made up of one or more *record types*. For example, a purchase order file might be made up of a record type PURCHASEORDER plus a second that is repeated for each item ordered on the purchase order – PURCHASEORDERITEM.
- **External interface file types** allow for output and input that might pass to and from other computer applications. Examples of this would be the transmission of accounting data from an order processing system to the main ledger system or the production of a file of direct debit details on a magnetic or electronic medium to be passed to the Bankers Automated Clearing System (BACS). Files shared among applications would also be counted here.
- **External inquiry types** – note the US spelling of inquiry – are transactions initiated by the user that provide information but do not update the internal files. The user inputs some information that directs the system to the details required.

The analyst has to identify each instance of each external user type in the projected system. Each component is then classified as having either high, average or low complexity. The counts of each external user type in each complexity band are multiplied by specified weights (see Table 5.2) to get FP scores, which are summed to obtain an overall FP count, which indicates the information processing size.

See A. J. Albrecht and J. E. Gaffney Jr., 'Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation' in M. Shepperd (ed.) *Software Engineering Metrics Volume 1*, McGraw-Hill, 1993.

Albrecht also specifies that outgoing external interface files should be double counted as logical internal file types as well.

The International FP User Group (IFPUG) have developed and published extensive rules governing FP counting. Hence Albrecht FPs are now often referred to as IFPUG FPs.

Table 5.2 Albrecht complexity multipliers

External user type	Multiplier		
	Low	Average	High
External input type	3	4	6
External output type	4	5	7
Logical internal file type	7	10	15
External interface file type	5	7	10
External inquiry type	3	4	6

Exercise 5.5

The task for which Brigette has been made responsible in Exercise 5.2 needs a program that will extract yearly salaries from the payroll file, and the details of courses and hours taught on each course by each member of staff from two files maintained by the time-tabling system. The program will calculate the staff costs for each course and put the results into a file that will then be read by the main accounting system. The program will also produce a report showing for each course the hours taught by each member of staff and the cost of those hours.

Using the method described above, calculate the Albrecht function points for this subsystem, assuming that the report is of high complexity, but that all the other elements are of average complexity.

One problem with FPs as originally defined by Albrecht was that the question of whether the external user type was of high, low or average complexity was rather subjective. The International FP User Group (IFPUG) have now promulgated rules on how this is to be judged. For example, in the case of logical internal files and external interface files, the boundaries shown in Table 5.3 are used to decide the complexity level.

Table 5.3 IFPUG file type complexity

Number of record types	Number of data types		
	<20	20 to 50	>50
1	low	low	average
2 to 5	low	average	high
> 5	average	high	high

Example 5.2

A logical internal file might contain data about purchase orders. These purchase orders might be organized into two separate record types: the main PURCHASEORDER details, namely purchase order number, supplier reference and purchase order date and then details for each PURCHASEORDERITEM specified in the order, namely the product code, the unit price and number ordered. The

number of record types for that file would therefore be 2 and the number of data types would be 6. According to Table 5.3, this file type would be rated as 'low'. This would mean that according to Table 5.2, the FP count would be seven for this file.

Tables 5.4 and 5.5 are used to allocate scores to external inputs and external outputs. Each external inquiry has to be counted both as if it were an external input and an external output and whichever score is higher is used.

Table 5.4 IFPUG External input complexity

<i>Number of file types accessed</i>	<i>Number of data types accessed</i>		
	<5	5 to 15	>15
0 or 1	low	low	average
2	low	average	high
> 2	average	high	high

Table 5.5 IFPUG External output complexity

<i>Number of file types</i>	<i>Number of data types</i>		
	<6	6 to 19	>19
0 or 1	low	low	average
2 or 3	low	average	high
> 3	average	high	high

Function point analysis now goes on to take into account the fact that the effort required to implement a computer-based information system will relate not just to the number and complexity of the features to be provided but also to the environment in which the system is to operate.

Fourteen factors have been identified that can influence the degree of difficulty associated with implementing a system. The list that Albrecht produced related particularly to the concerns of information system developers in the late 1970s and early 1980s. Some technology that was then new and relatively threatening is now well established.

The technical complexity adjustment (TCA) calculation has had lots of problems. Some have even found that it produces less accurate estimates than using the unadjusted function point count. Because of these difficulties, we are going to omit further discussion of the TCA.

Tables have been calculated to convert the FPs to lines of code for various languages. For example, it is suggested that 91 lines of Cobol are needed on average to implement an FP, while for C the figure is 128 and for QuickBasic is 64.

Further details on TCA can be found in the Albrecht and Gaffney paper.

Some of these conversion factors do not seem to be very convincing (e.g. 6 SLOC per function point for spreadsheets).

Exercise 5.6

In the case of the subsystem described in Exercise 5.5 for which Brigette is responsible at Brightmouth College, how many lines of Cobol code should be needed to implement this subsystem, according to the standard conversion?

5.9 Function points Mark II

This method has come into the public domain with the publication of the book by Charles R. Symons *Software Sizing and Estimating – Mark II FPA*, John Wiley and Sons, 1991.

The Mark II method has been recommended by the CCTA (Central Computer and Telecommunications Agency), which lays down standards for UK government projects. At one time this Mark II approach seemed to be a good method to use with SSADM but some difficulties are now apparent. The 'Mark II' label implies an improvement and replacement of the Albrecht method. The Albrecht (now IFPUG) method, however, has had many refinements made to it and FPA Mark II remains a minority method used mainly in the UK.

As with Albrecht, the information processing size is initially measured in unadjusted function points (UFPs) to which a technical complexity adjustment can then be applied (TCA). The assumption here is that an information system comprises transactions that have the basic structure shown in Figure 5.2.

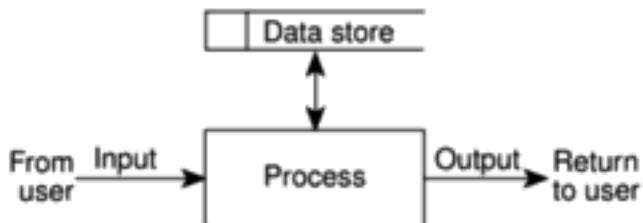


Figure 5.2 Model of a transaction.

For each transaction the UFPs are calculated:

$$\begin{aligned}
 W_i \times (\text{number of input data element types}) + \\
 W_e \times (\text{number of entity types referenced}) + \\
 W_o \times (\text{number of output data element types})
 \end{aligned}$$

Here, W_i , W_e , and W_o are weightings that might be derived by asking developers what proportion of effort has been spent in previous projects developing those parts of the software that deal with processing inputs, accessing and modifying stored data and processing outputs. From this it should be possible to work out the average hours of work generated by instances of each type of element.

The averages are then normalized into ratios, or weightings, which add up to 2.5. If this way of getting hold of the weightings seems too time-consuming then some industry averages are available, which are currently (1999) 0.58 for W_i , 1.66 for W_e and 0.26 for W_o .

The only reason why 2.5 was adopted here was to produce FP counts similar to the Albrecht equivalents.

A cash receipt transaction in the IOE maintenance accounts system accesses two entity types – INVOICE and CASHRECEIPT.

Example 5.3

The data elements that are input are:

- InvoiceNumber
- DateReceived
- CashReceived

If an INVOICE record is not found for the invoice number, then an error message is issued. If the invoice number is found, then a CASHRECEIPT record is created.

The error message constitutes the only output data element that the transaction has to cater for.

The unadjusted function points, using the industry average weightings, for this transaction would therefore be:

$$(0.58 \times 3) + (1.66 \times 2) + (0.26 \times 1) = 5.32$$

One of the transactions that will be part of the IOE maintenance group accounts subsystem for which Amanda is responsible will be used to set up details of new group account customers.

Exercise 5.7

The operator will input:

- CustomerAccountNumber
- CustomerName
- Address
- Postcode
- CustomerType
- StatementProductionDate

All this information will be set up in a CUSTOMER record on the system's database. If a CUSTOMER account already exists for the account number that has been input, an error message will be displayed to the operator.

Calculate the number of unadjusted Mark II function points for this transaction, using the industry average weightings.

Mark II FPs follow the Albrecht method in recognizing that one system delivering the same functionality as another might be more difficult to implement (but also more valuable to the users) because of additional technical requirements. For example, the incorporation of additional security measures would increase the amount of effort to deliver the system. The original Albrecht FP method identified

14 technical complexity adjustment factors – Mark II FPs identify five more factors:

- interfaces to other applications;
- special security features;
- direct access for third parties;
- user training features;
- documentation requirements.

The addition of other factors to suit local circumstances is encouraged.

With both the Albrecht and Symons methods, FPs can be counted for previous projects where actual effort is known. If you have figures for the effort expended on past projects (in work-days for instance) and also the system sizes in FPs, you should be able to work out a productivity rate, that is:

$$\text{productivity} = \text{size}/\text{effort}$$

For new projects, the function points can be counted and then the effort can be projected using the productivity rate derived above:

$$\text{effort} = \text{size}/\text{productivity}$$

A more sophisticated way of doing this would be by using the statistical technique, least squares regression, to derive an equation in the form:

$$\text{effort} = \text{constant}_1 + \text{size} \times \text{constant}_2$$

Symons is very much against the idea of using function points to estimate SLOC rather than effort. One finding by Symons is that productivity, that is, the effort per function point to implement a system, is influenced very much by the size of the project. In general, larger projects, up to a certain point, tend to be more productive because of economies of scale. However, beyond a certain size they tend to become less productive because of additional management overheads.

Some of the rules and weightings used in FP counting, especially in the case of the Albrecht flavour, are rather arbitrary and have been criticized by academic writers on this account. FPs, however, are widely used in practice because of the lack of other methods of gauging the functional size of information systems.

This simplified approach assumes rather unrealistically that the factors affecting productivity are the same for each project.

See R. D. Banker, R. Kauffman and R. Kumar, 'An Empirical Test of Object-based Output Measurement Methods' Journal of MIS, 8(3), 1993.

5.10 Object points

This approach was devised at the Leonard N. Stern School of Business, New York University. It has similarities with the FP approach, but takes account of features that might be more readily identifiable if you are building a system using a high-level application building tool. The reader should be warned that despite its name,

the technique has no direct bearing on object-oriented techniques. The approach uses counts of the screens, reports and 3GL components that an application might possess – it is these that are referred to as *objects*. Each object has to be classified as one of the following:

- simple;
- medium;
- difficult.

Table 5.6 and Table 5.7 show the scheme used to make this classification. The numbers of objects at each level are multiplied by the appropriate complexity weighting shown in Table 5.8. The weighted sub-totals are then summed to get an overall score for the application.

In other development environments, other types of object and other weightings would be more appropriate.

Table 5.6 Object Points for screens

Number of views contained	Number and source of data tables		
	Total < 4 (<2 servers <3 clients)	Total < 8 (<3 servers 3 to 5 clients)	Total > 7 (>3 servers > 5 clients)
< 3	simple	simple	medium
3 to 7	simple	medium	difficult
> 7	medium	difficult	difficult

Further details can be found in R. Kauffman and R. Kumar's report 'Modelling Estimation Expertise in Object Based ICASE Environments', Stern School of Business, 1993.

Table 5.7 Object Points for reports

Number of sections contained	Number and source of data tables		
	Total < 4 (<2 servers <3 clients)	Total < 8 (<3 servers 3 to 5 clients)	Total > 7 (>3 servers > 5 clients)
< 2	simple	simple	medium
2 or 3	simple	medium	difficult
> 3	medium	difficult	difficult

Table 5.8 Object Points complexity weightings

Object type	Complexity weighting		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	—	—	10

Some of these objects might not need to be developed as there are already existing components that can be utilized. The object point score can be adjusted to take this into account. Say that in an application containing 840 object points, 20% can be supplied by using existing components, then the adjusted new object points (NOP) score would be:

$$NOP = 840 \times (100 - 20)/100 = 672$$

Finally a productivity rate (PROD) has to be identified. It would be best if the estimator could use details of past projects to derive this. As an example, the developers of object points have published the details in Table 5.9 to calculate PROD. In the situation where this information was gathered, as the CASE tool's features were improving with successive releases, so the experience of the developers with the tool was growing too.

Table 5.9 Object point effort conversion

Developer's experience and capability/ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

An estimate of the person-months needed to carry out the project is then calculated by dividing PROD into NOP. For example, given the 672 new object points above and a development environment where productivity was nominal, then the estimated effort for the project would be $672/13 = 52$ months.

5.11 A procedural code-oriented approach

The previous approach would be useful at the design stage of a project and where a procedural programming language is not the primary vehicle for development. However how could you estimate the effort to develop an individual software module using a procedural language? An approach might be based on the following steps.

1. Envisage the number and type of programs in the final system

This is easiest where the system is of a conventional and well understood nature. Most information systems are built from a small set of system operations, such as insert, amend, update, display, delete, print. The same principle should equally apply to embedded systems, albeit with a different set of primitive functions.

2. Estimate the SLOC of each identified program

The estimator must have a particular implementation language in mind for this step.

One way to judge the number of instructions likely to be in a program is to draw up a program structure diagram and to visualize how many instructions would be needed to implement each identified procedure. The estimator might look at existing programs that have a similar functional description to help in this process.

Where programs for an information system are similar (for instance, they are data validation programs) then the number of data item types processed by each program is likely to be the major influence on size.

3. Estimate the work content, taking into account complexity and technical difficulty

The practice is to multiply the SLOC estimate by a factor for complexity and technical difficulty. This factor will depend largely on the subjective judgement of the estimator. For example, the requirement to meet particular highly constrained performance targets can greatly increase programming effort.

A weighting can be given when there is uncertainty, for example about a new technique used in particular module, but this should not be excessive. Where there is a large amount of uncertainty then specific measures should be taken to reduce this by such means as the use of exploratory prototypes.

4. Calculate the work-days effort

Historical data can be used to provide ratios to convert weighted SLOC to effort. These conversion factors are often based on the productivity of a 'standard programmer' of about 15–18 months of experience. In installations where the rate of turnover is lower and the average programmer experience is higher this might be reflected in the conversion rate employed.

Note that the steps above can be used to derive an estimate of lines of code that can be used as an input to one of the COCOMO models, which are now about to be described.

Draw up an outline program structure diagram for a program to do the processing described in Exercise 5.7, which sets up CUSTOMER records. For each box on your diagram, estimate the number of lines of code needed to implement the routine in a third generation language such as Cobol.

Function point analysis
Mark II is also based on the idea that the number of data item types processed influences program size.

See Chapter 4 for a discussion of prototypes.

Exercise 5.8

5.12 COCOMO: a parametric model

Boehm's COCOMO (COnstructive COst MOdel) is often referred to in the literature on software project management, particularly in connection with software estimating. The term COCOMO really refers to a group of models.

Boehm originally based his models in the late 1970s on a study of 63 projects. Of these only seven were business systems and so they could be used with applications other than information systems. The basic model was built around the equation

Because there is now a newer COCOMO II, the older version is now referred to as COCOMO 81.

Boehm originally used mm (for man-months) when he wrote *Software Engineering Economics*.

Generally, information systems were regarded as organic while real-time systems were embedded.

$$\text{effort} = c \times \text{size}^k$$

where effort is measured in pm , or the number of ‘person-months’ consisting of units of 152 working hours, size is measured in $kdsi$, thousands of delivered source code instructions, and c and k are constants.

The first step was to derive an estimate of the system size in terms of $kdsi$. The constants, c and k (see Table 5.10), depended on whether the system could be classified, in Boehm’s terms, as ‘organic’, ‘semi-detached’ or ‘embedded’. These related to the technical nature of the system and the development environment.

- **Organic mode** – this would typically be the case when relatively small teams developed software in a highly familiar in-house environment and when the system being developed was small and the interface requirements were flexible.
- **Embedded mode** – this meant the product being developed had to operate within very tight constraints and changes to the system were very costly.
- **Semi-detached mode** – this combined elements of the organic and the embedded modes or had characteristics that came between the two.

Table 5.10 COCOMO constants

System type	c	k
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

The exponent value k , when it is greater than 1, means that larger projects are seen as requiring disproportionately more effort than smaller ones. This reflected Boehm’s finding that larger projects tended to be less productive than smaller ones because they needed more effort for management and co-ordination.

Exercise 5.9

Apply the basic COCOMO model to the lines of code figures in Table 5.1 to generate estimated work-months of effort, assuming an organic mode. Compare the calculated figures with the actuals.

As well as the intermediate model, a further, detailed, COCOMO model attempts to allocate effort to individual project phases.

Boehm in fact found this, by itself, to be a poor predictor of the effort required and so went on to develop the intermediate version of COCOMO, which took into account 15 cost drivers. In the intermediate model, a nominal effort estimate, (pm_{nom}) is derived in a similar way as for the basic model.

The nominal estimate is then adjusted by a development effort multiplier (dem):

$$pm_{est} = pm_{nom} \times dem$$

where dem is calculated by taking into account multipliers based on the effort drivers in Table 5.11.

Table 5.11 COCOMO81 intermediate cost drivers

Driver type	Code	Cost driver
Product attributes	RELY	required software reliability
	DATA	database size
	CPLX	product complexity
Computer attributes	TIME	execution time constraints
	STOR	main storage constraints
	VIRT	virtual machine volatility – degree to which the operating system changes
	TURN	computer turn around time
Personnel attributes	ACAP	analyst capability
	AEXP	application experience
	PCAP	programmer capability
	VEXP	virtual machine (i.e. operating system) experience
	LEXP	programming language experience
Project attributes	MODP	use of modern programming practices
	TOOL	use of software tools
	SCED	required development schedule.

These multipliers take into account such influences on productivity as Boehm's suggestion that having a programming team fully conversant with the programming language to be used could reduce the effort required to implement the project by up to 20% compared to a team with a very low or initially non-existent familiarity with the programming language. In fact, the biggest influence on productivity according to Boehm is the capability of the implementation team.

As an example of the approach, an organization might decide to use the following multipliers for assessing the effect of analyst capability (ACAP).

Very low	1.46
Low	1.19
Nominal	1.00
High	0.80
Very High	0.71

If the analysts involved in a project, taken as a whole, generally possess above average talent and productivity then the estimator might rate the ACAP as high and use a multiplier of 0.8, effectively reducing the nominal estimate by 20%.

The overall dem is calculated by multiplying the multipliers selected for each cost driver in Table 5.11 to create a single combined multiplier.

Exercise 5.10

At IOE, most of the systems that are developed are technically similar, so that the product, computer and project attributes, as listed in Table 5.11 do not change from one project to another and are given a unit multiplier of 1.0. Only personnel attributes differ and the following table is used by the organization to take this into account.

Attribute	Very low	Low	Nominal	High	Very high
ACAP	1.46	1.19	1.00	0.86	0.71
AEXP	1.29	1.13	1.00	0.91	0.82
PCAP	1.42	1.17	1.00	0.80	0.70
VEXP	1.21	1.10	1.00	0.90	—
LEXP	1.14	1.07	1.00	0.95	—

On the new IOE group maintenance accounts project, the analyst is regarded as being of exceptionally high quality. The programmers are of high quality but have little experience of the particular application area and are going to use a programming language that is new to them. They are however familiar with the operating system environment and thus can be rated as high on VEXP.

What would the *dem* for this project? If the nominal estimate for this project was four person-months, what would be the final estimate?

The detailed *COCOMO II Model Definition Manual* has been published by the Centre for Software Engineering, University of Southern California.

A new family of models, COCOMO II, is currently (1999) being refined by Barry Boehm and his co-workers. This approach uses various multipliers and exponents, the values of which have been set initially by experts. However, a database containing the performance details of executed projects is being built up and periodically analysed so that the expert judgements can be progressively replaced by values derived from actual projects. The new models take into account that there is now a wider range of process models in common use for software development projects than in the late 1970s and early 1980s. As we noted earlier, estimates are required at different stages in the system life cycle and COCOMO II has been designed to accommodate this by having models for three different stages.

Application composition Where the external features of the system that the users will experience are designed. Prototyping will typically be employed to do this. With small applications that can be built using high-productivity application-building tools, development can stop at this point.

Early design Where the fundamental software structures are designed. With larger, more demanding systems, where, for example, there will be large volumes of transactions and performance is important, careful attention will need to be paid to the architecture to be adopted.

Post architecture Where the software structures undergo final construction, modification and tuning to create a system that will perform as required.

To estimate the effort for *application composition*, the counting of object points, which were described earlier, is recommended by the developers of COCOMO II.

At the *early design* stage, FPs are recommended as the way of gauging a basic system size. An FP count might be converted to a SLOC equivalent by multiplying the FPs by a factor for the programming language that is to be used.

The following model can then be used to calculate an estimate of person-months.

$$pm = A \times size^sf \times em_1 \times em_2 \times \dots \times em_n$$

Note that where
COCOMO 81 used 'man-
months', COCOMO II
uses 'person-months'.

Where pm is the effort in 'person-months', A is a constant (in 1998 it was 2.45), size is measured in SLOC (which might have been derived from an FP count as explained above), and sf is exponent scale factor.

The scale factor is derived thus:

$$sf = 1.01 + 0.01 \times \sum(\text{exponent driver ratings})$$

What is the maximum value that the scale factor (sf) can have, given that there are five exponent drivers and the maximum rating for an individual driver is five and the minimum is zero?

Exercise 5.11

The qualities that govern the exponent drivers used to calculate the scale factor are listed below. Note that the less each quality is applicable, the bigger the value given to the exponent driver. The fact that these factors are used to calculate an exponent implies that the *lack* of these qualities will *increase* the effort required disproportionately more on larger projects.

- **Precededness** This quality refers to the degree to which there are *precedents*, similar cases in the past, for the project that is being planned. The greater the novelty of the new system, the more uncertainty there is and the higher the value given to the exponent driver.
- **Development flexibility** This is the degree to which the requirements can be met in a number of different ways. The less flexibility there is, the higher the value of the exponent driver.
- **Architecture/risk resolution** This relates to the degree of uncertainty there is about the requirements. If they are not firmly fixed and are liable to change then this would lead to a high value being given to this exponent driver.
- **Team cohesion** This reflects the degree to which there is a large dispersed team (perhaps in several countries) as opposed to there being a small tightly knit team.

- **Process maturity** The chapter on software quality explains the process maturity model. The more structured and organized the way the software is produced, the lower uncertainty and the lower the rating will be for this exponent driver.

Exercise 5.12

A new project has ‘average’ novelty for the software house that is going to execute it and is thus given a 3 rating on this account for precedentedness. Development flexibility is high to the extent that this generates a zero rating, but requirements might change radically and so the risk resolution exponent is rated at 4. The team is very cohesive and this generates a rating of 1, but the software house as a whole tends to be very informal in its standards and procedures and the process maturity driver has therefore been given a value of 4.

What would be the scale factor, sf , that would applicable in this case?

In the COCOMO II model the *effort multipliers*, em , are similar in nature to the development effort multipliers, dem , used in the original COCOMO. There are seven of these multipliers that are relevant to early design and sixteen that can be used at the post architecture stage. Table 5.12 lists the effort multipliers for early design and Table 5.13 for post architecture. As with COCOMO 81, each of these multipliers might, for a particular application, be given a rating of very low, low, nominal, high or very high. Each rating for each effort multiplier has a value associated with it. A value greater than 1 means that development effort is increased, while a value less than 1 causes effort to be decreased. The nominal rating means that the multiplier has no effect on the estimate, that is, it is 1. The intention is that the ratings that these and other values use in COCOMO II will be modified and refined over time as details of actual projects are added to the database.

Table 5.12 COCOMOII Early Design effort multipliers

<i>Code</i>	<i>Effort modifier</i>
RCPX	Product reliability and complexity
RUSE	Required reusability
PDIF	Platform difficulty
PERS	Personnel capability
PREX	Personnel experience
FCIL	Facilities available
SCED	Schedule pressure

Table 5.13 COCOMOII Post Architecture effort multipliers

<i>Modifier type</i>	<i>Code</i>	<i>Effort modifier</i>
Product attributes	RELY	Required software reliability
	DATA	Database size
	DOCU	Documentation match to life-cycle needs
	CPLX	Product complexity
	REUSE	Required reusability
Platform attributes	TIME	Execution time constraint
	STOR	Main storage constraint
	PVOL	Platform volatility
Personnel attributes	ACAP	Analyst capabilities
	AEXP	Application experience
	PCAP	Programmer capabilities
	PEXP	Platform experience
	LEXP	Programming language experience
	PCON	Personnel continuity
Project attributes	TOOL	Use of software tools
	SITE	Multisite development

5.13 Conclusion

To summarize some key points:

- estimates are really management targets;
- collect as much information about previous projects as possible;
- use more than one method of estimating;
- top-down approaches will be used at the earlier stages of project planning while bottom-up approaches will be more prominent later on;
- be careful about using other people's historical productivity data as a basis for your estimates, especially if it comes from a different environment (this includes COCOMO!);
- seek a range of opinions;
- document your method of doing estimates and record all your assumptions.

5.14 Additional exercises

- Identify primary and secondary effort drivers for the following activities:
 - installing computer workstations in a new office;
 - transporting assembled personal computers from the factory where they were assembled to warehouses distributed in different parts of the country;
 - typing in and checking the correctness of data that is populating a new database;
 - system testing a newly written software application.
- If you were asked as an expert to provide an estimate of the effort needed to make certain changes to an existing piece of software, what information would you like to have to hand to assist you in making that estimate?
- Consider the details held about previously developed software modules shown in Table 5.14. A new module has seven inputs, one entity type access and seven outputs. Which of the modules *a* to *e* is the closest analogy in terms of Euclidean distance?

Table 5.14 Data concerning previously developed modules

module	inputs	entity types accessed	outputs	days
a	1	2	10	2.60
b	10	2	1	3.90
c	5	1	1	1.83
d	2	3	11	3.50
e	1	3	20	4.30

- Using the data in Table 5.14, calculate the Simons Mark II FPs for each module. Using the results, calculate the effort needed for the new module described in Question 3. How does this estimate compare to the one based on analogy?
- A report in a college time-tabling system produces a report showing the students who should be attending each time-tabled teaching activity. Four files are accessed: the STAFF file, the STUDENT file, the STUDENTOPTION file and the TEACHINGACTIVITY file. The report contains the following information:

For each teaching activity:

Teaching Activity Reference
Topic Name
Staff Forename
Staff Surname
Title

Semester (1 or 2)

Day Of Week

Time

Duration

Location

For each student:

Student Forename

Student Surname

Calculate the both IFPUG and Mark II FPs that this transaction would generate.

Can you identify the factors that would tend to make the two methods generate divergent counts?

Chapter 6

Activity planning

OBJECTIVES

When you have completed this chapter you will be able to:

- produce an activity plan for a project;
 - estimate the overall duration of a project;
 - create a critical path and a precedence network for a project.
-

6.1 Introduction

In earlier chapters we looked at methods for forecasting the effort required for a project – both for the project as a whole and for individual activities. A detailed plan for the project, however, must also include a schedule indicating the start and completion times for each activity. This will enable us to:

- ensure that the appropriate resources will be available precisely when required;
- avoid different activities competing for the same resources at the same time;
- produce a detailed schedule showing which staff carry out each activity;
- produce a detailed plan against which actual achievement may be measured;
- produce a timed cash flow forecast;
- replan the project during its life to correct drift from the target.

To be effective, a plan must be stated as a set of targets, the achievement or non-achievement of which can be unambiguously measured. The activity plan does this by providing a target start and completion date for each activity (or a window within which each activity may be carried out). The starts and completions of activities must be clearly visible and this is one of the reasons why it is advisable to ensure that each and every project activity produces some tangible product or

Project monitoring is discussed in more detail in Chapter 9.

'deliverable'. Monitoring the project's progress is then, at least in part, a case of ensuring that the products of each activity are delivered on time.

As a project progresses it is unlikely that everything will go according to plan. Much of the job of project management concerns recognizing when something has gone wrong, identifying its causes and revising the plan to mitigate its effects. The activity plan should provide a means of evaluating the consequences of not meeting any of the activity target dates and guidance as to how the plan might most effectively be modified to bring the project back to target. We shall see that the activity plan may well also offer guidance as to which components of a project should be most closely monitored.

6.2 The objectives of activity planning

In addition to providing project and resource schedules, activity planning aims to achieve a number of other objectives which may be summarized as follows.

- **Feasibility assessment** Is the project possible within required timescales and resource constraints? It is not until we have constructed a detailed plan that we can forecast a completion date with any reasonable knowledge of its achievability. The fact that a project may have been estimated as requiring two work-years effort might not mean that it would be feasible to complete it within, say, three months were eight people to work on it – that will depend upon the availability of staff and the degree to which activities may be undertaken in parallel.
- **Resource allocation** What are the most effective ways of allocating resources to the project and when should they be available? The project plan allows us to investigate the relationship between timescales and resource availability (in general, allocating additional resources to a project shortens its duration) and the efficacy of additional spending on resource procurement.
- **Detailed costing** How much will the project cost and when is that expenditure likely to take place? After producing an activity plan and allocating specific resources, we can obtain more detailed estimates of costs and their timing.
- **Motivation** Providing targets and being seen to monitor achievement against targets is an effective way of motivating staff, particularly where they have been involved in setting those targets in the first place.
- **Co-ordination** When do the staff in different departments need to be available to work on a particular project and when do staff need to be transferred between projects? The project plan, particularly with large projects involving more than a single project team, provides an effective vehicle for communication and co-ordination among teams. In situations where staff may need to be transferred between project teams (or work concurrently on more

Chapter 11 discusses motivation in more detail.

This co-ordination will normally form part of Programme Management.

than one project), a set of integrated project schedules should ensure that such staff are available when required and do not suffer periods of enforced idleness.

Activity planning and scheduling techniques place an emphasis on completing the project in a minimum time at an acceptable cost or, alternatively, meeting an arbitrarily set target date at minimum cost. These are not, in themselves, concerned with meeting quality targets, which generally impose constraints on the scheduling process.

One effective way of shortening project durations is to carry out activities in parallel. Clearly we cannot undertake all the activities at the same time – some require the completion of others before they can start and there are likely to be resource constraints limiting how much may be done simultaneously. Activity scheduling will, however, give us an indication of the cost of these constraints in terms of lengthening timescales and provide us with an indication of how timescales may be shortened by relaxing those constraints. It is up to us, if we try relaxing precedence constraints by, for example, allowing a program coding task to commence before the design has been completed, to ensure that we are clear about the potential effects on product quality

In Chapter 2 we saw that Amanda's wish to check that four module specifications were correct, while increasing the likely quality of the product, created a constraint that could potentially delay the next stage of the project.

6.3 When to plan

Planning is an ongoing process of refinement, each iteration becoming more detailed and more accurate than the last. Over successive iterations, the emphasis and purpose of planning will shift.

During the feasibility study and project start-up, the main purpose of planning will be to estimate timescales and the risks of not achieving target completion dates or keeping within budget. As the project proceeds beyond the feasibility study, the emphasis will be placed upon the production of activity plans for ensuring resource availability and cash flow control.

Throughout the project, until the final deliverable has reached the customer, monitoring and replanning must continue to correct any drift that might prevent meeting time or cost targets.

6.4 Project schedules

Before work commences on a project or, possibly, a stage of a larger project, the project plan must be developed to the level of showing dates when each activity should start and finish and when and how much of each resource will be required. Once the plan has been refined to this level of detail we call it a *project schedule*. Creating a project schedule comprises four main stages.

The first step in producing the plan is to decide what activities need to be carried out and in what order they are to be done. From this we can construct an *ideal activity plan* – that is, a plan of when each activity would ideally be undertaken were resources not a constraint. It is the creation of the ideal activity plan that we shall discuss in this chapter. This activity plan is generated by Steps 4 and 5 of Step Wise (Figure 6.1).

On a large project, detailed plans for the later stages will be delayed until information about the work required has emerged from the earlier stages.

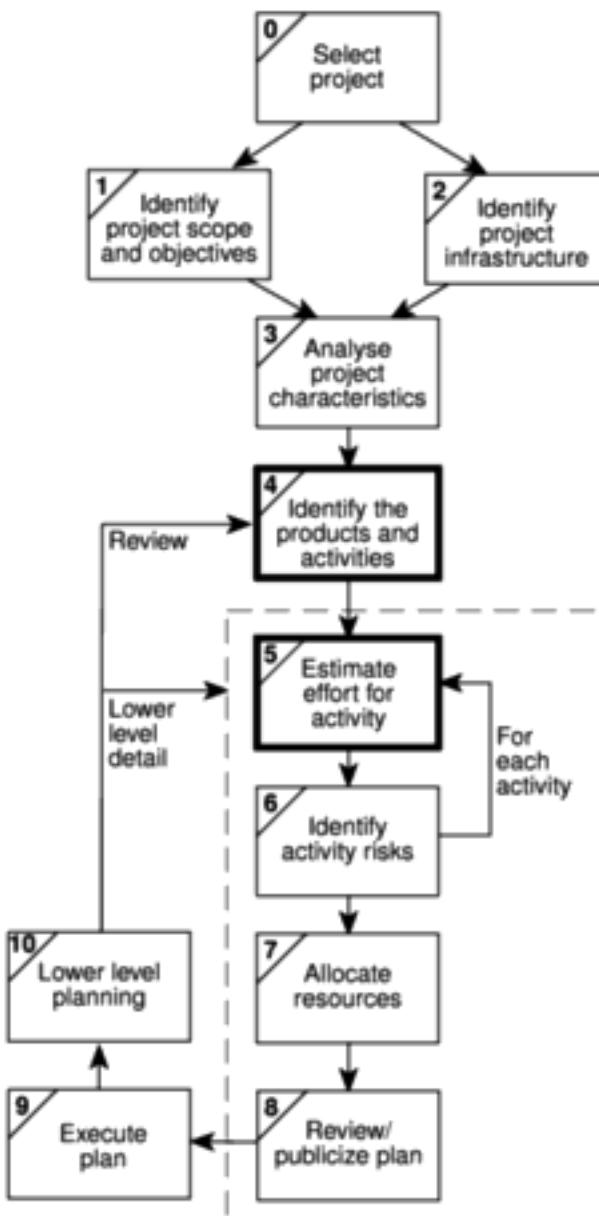


Figure 6.1 Activity planning is carried out in Steps 4 and 5.

The ideal activity plan will then be the subject of an activity risk analysis, aimed at identifying potential problems. This might suggest alterations to the ideal activity plan and will almost certainly have implications for resource allocation. Activity risk analysis is the subject of Chapter 7.

The third step is resource allocation. The expected availability of resources might place constraints on when certain activities can be carried out, and our ideal plan might need to be adapted to take account of this. Resource allocation is covered in Chapter 8.

The final step is *schedule production*. Once resources have been allocated to each activity, we will be in a position to draw up and publish a project schedule, which indicates planned start and completion dates and a resource requirements statement for each activity. Chapter 9 discusses how this is done and the role of the schedule in managing a project.

6.5 Projects and activities

Defining activities

Before we try to identify the activities that make up a project it is worth reviewing what we mean by a project and its activities and adding some assumptions that will be relevant when we start to produce an activity plan.

- a project is composed of a number of inter-related activities;
- a project may start when at least one of its activities is ready to start;
- a project will be completed when all of the activities it encompasses have been completed;
- an activity must have a clearly defined start and a clearly defined end-point, normally marked by the production of a tangible deliverable;
- if an activity requires a resource (as most do) then that resource requirement must be forecastable and is assumed to be required at a constant level throughout the duration of the activity;
- the duration of an activity must be forecastable – assuming normal circumstances, and the reasonable availability of resources;
- some activities might require that others are completed before they can begin (these are known as *precedence requirements*).

Activities must be defined so that they meet these criteria. Any activity that does not meet these criteria must be redefined.

Identifying activities

Essentially there are three approaches to identifying the activities or tasks that make up a project – we shall call them the *activity-based approach*, the *product-based approach* and the *hybrid approach*.

The activity-based approach The activity-based approach consists of creating a list of all the activities that the project is thought to involve. This might involve a brainstorming session involving the whole project team or it might stem from an analysis of similar past projects. When listing activities, particularly for a large project, it might be helpful to subdivide the project into the main life style stages and consider each of these separately.

Rather than doing this in an ad hoc manner, with the obvious risks of omitting or double-counting tasks, a much favoured way of generating a task list is to create a *Work Breakdown Structure* (WBS). This involves identifying the main (or high-level) tasks required to complete a project and then breaking each of these down into a set of lower-level tasks. Figure 6.2 shows a fragment of a WBS where the design task has been broken down into three tasks and one of these has been further decomposed into two tasks.

Activities are added to a branch in the structure if they directly contribute to the task immediately above – if they do not contribute to the parent task, then they should not be added to that branch. The tasks at each level in any branch should

WBSs are advocated by BS 6079, which is discussed in Appendix B.

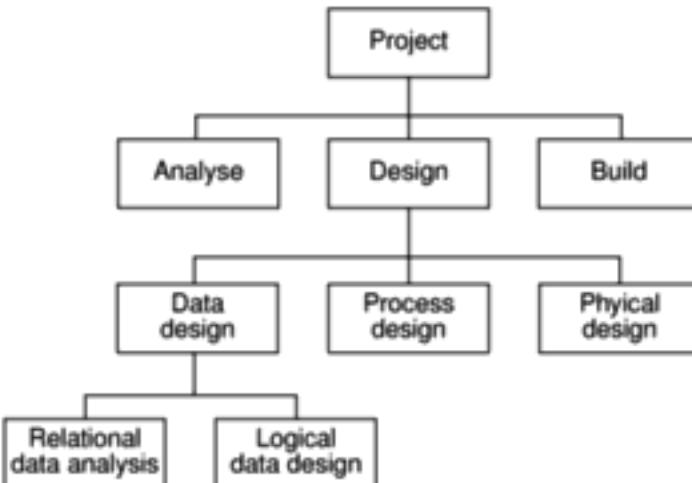


Figure 6.2 A fragment of an activity-based Work Breakdown Structure.

include everything that is required to complete the task at the higher level – if they are not a comprehensive definition of the parent task, then something is missing.

When preparing a WBS, consideration must be given to the final level of detail or depth of the structure. Too great a depth will result in a large number of small tasks that will be difficult to manage, whereas a too shallow structure will provide insufficient detail for project control. Each branch should, however, be broken down at least to a level where each leaf may be assigned to an individual or responsible section within the organization.

Advantages claimed for the WBS approach include the belief that it is much more likely to result in a task catalogue that is complete and is composed of non-overlapping activities. Note that it is only the leaves of the structure that comprise the list of activities comprising the project – higher-level nodes merely represent collections of activities.

The WBS also represents a structure that may be refined as the project proceeds. In the early part of a project we might use a relatively high-level or shallow WBS, which can be developed as information becomes available, typically during the project's analysis and specification phases.

Once the project's activities have been identified (whether or not by using a WBS) they need to be sequenced in the sense of deciding which activities need to be completed before others can start.

The product-based approach The product-based approach, used in PRINCE 2 and Step Wise, has already been described in Chapter 2. It consists of producing a Product Breakdown Structure and a Product Flow Diagram. The PFD indicates, for each product, which other products are required as inputs. The PFD can therefore be easily transformed into an ordered list of activities by identifying the transformations that turn some products into others. Proponents of this approach claim that it is less likely that a product will be left out of a PBS than that an activity might be omitted from an unstructured activity list.

A complete task catalogue will normally include task definitions along with task input and output products and other task-related information.

This approach is particularly appropriate if using a methodology such as SSADM, which clearly specifies, for each step or task, each of the products required and the activities required to produce it. The SSADM Reference Manual provides a set of generic PBSs for each stage in SSADM (such as that shown in Figure 6.3), which can be used as a basis for generating a project-specific PBS.

Most good texts on SSADM will explain the tailoring of the generic PBSs and Activity Networks. The illustrations here are taken from M. Goodland and C. Slater, *SSADM Version 4: A Practical Approach*, McGraw-Hill, 1995.



Figure 6.3 SSADM Product Breakdown Structure for Requirements Specification (adapted from Goodland and Slater).

The SSADM Reference Manual also supplies generic activity networks and, using the project-specific PBS and derived PFD, these may be used as a basis for developing a project-specific activity network. Figure 6.4 illustrates an activity network for the activities required to create the products in Figure 6.3.

Notice how the development of a PFD leads directly to an activity network that indicates the sequencing of activities – in Figure 6.4, activity 340 (Enhance required data model) requires products from activity 330 and activity 360 needs products from both activities 330 and 340.

The activity numbers in Figure 6.4 are the step numbers used by SSADM version 4.

The hybrid approach The WBS illustrated in Figure 6.2 is based entirely on a structuring of activities. Alternatively, and perhaps more commonly, a WBS may be based upon the project's products as illustrated in Figure 6.5, which is in turn based on a simple list of final deliverables and, for each deliverable, a set of activities required to produce that product. Figure 6.5 illustrates a flat WBS and it is likely that, in a project of any size, it would be beneficial to introduce additional levels – structuring both products and activities. The degree to which the structuring is product-based or activity-based might be influenced by the nature of the project and the particular development method adopted. As with a purely activity-based WBS, having identified the activities we are then left with the task of sequencing them.

BS 6079 states that WBSs may be product-based, cost-centre-based, task-based or function-based but states that product-based WBSs are preferred.

Not all of the products in this activity structuring will be final products. Some will be further refined in subsequent steps.

The current version of the SSADM reference manual provides generic PBSs and generic activity networks, but not generic PFDs. Stress is placed on the fact that these are generic structures and it is important that project-specific versions are created. A project-specific PFD would be produced as part of this process.

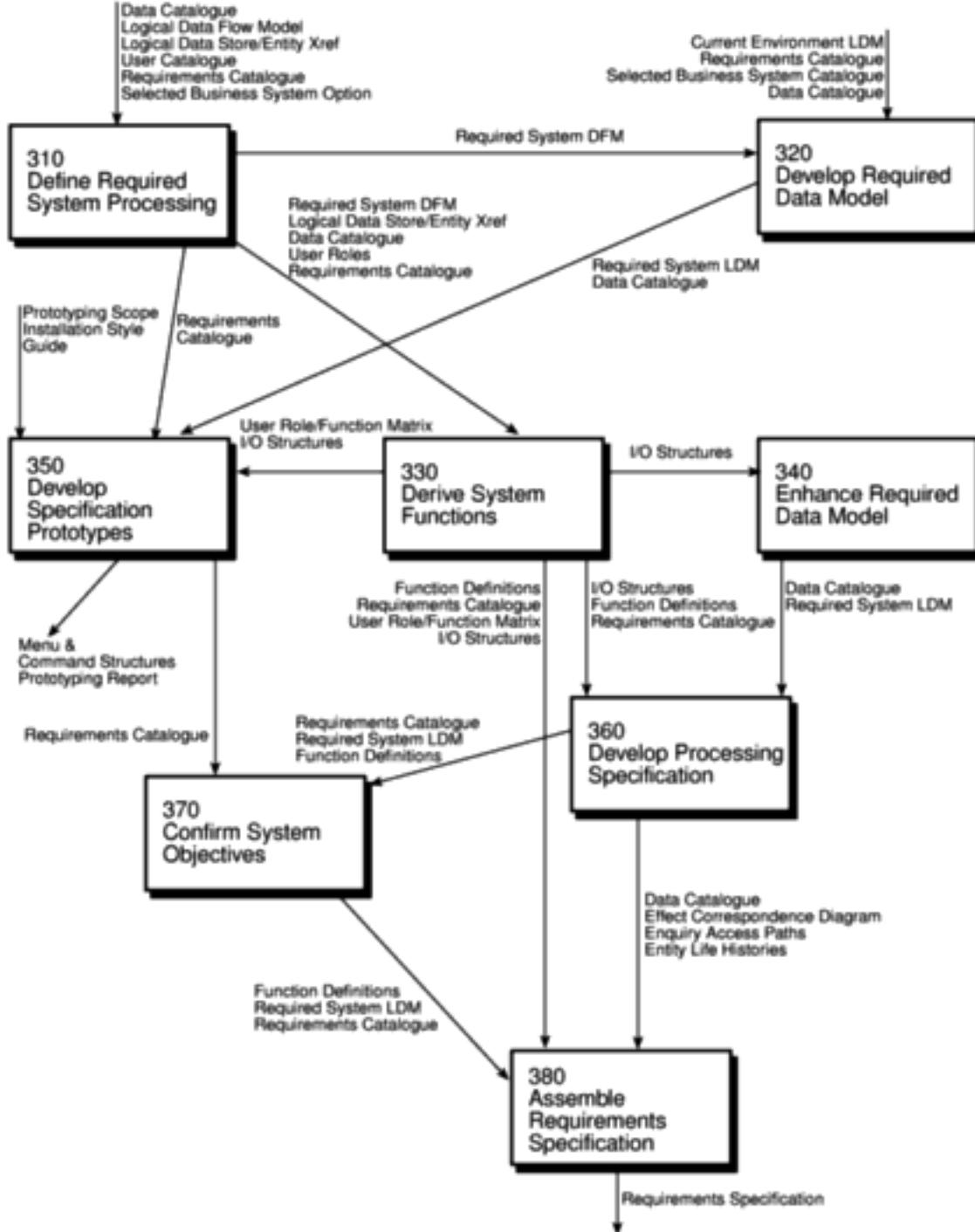


Figure 6.4 A structuring of activities for the SSADM Requirements Specification stage (from Goodland and Slater).

A framework dictating the number of levels and the nature of each level in the structure may be imposed on a WBS. For example, in their MITP methodology, IBM recommend that the following five levels should be used in a WBS:

- **Level 1: Project.**
- **Level 2: Deliverables** such as software, manuals and training courses.

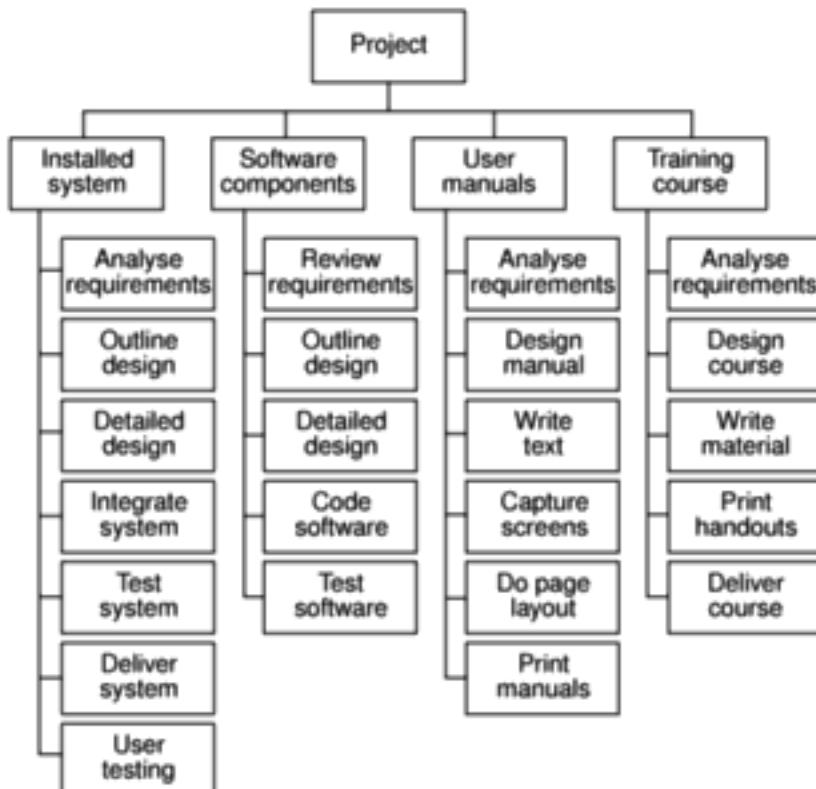


Figure 6.5 A Work Breakdown Structure based on deliverables.

- **Level 3: Components** which are the key work items needed to produce deliverables, such as the modules and tests required to produce the system software.
- **Level 4: Work-packages** which are major work items, or collections of related tasks, required to produce a component.
- **Level 5: Tasks** which are tasks that will normally be the responsibility of a single person.

6.6 Sequencing and scheduling activities

Throughout a project, we will require a schedule that clearly indicates when each of the project's activities is planned to occur and what resources it will need. We shall be considering scheduling in more detail in Chapter 8, but let us consider in outline how we might present a schedule for a small project. One way of presenting such a plan is to use a bar chart as shown in Figure 6.6.

The chart shown has been drawn up taking account of the nature of the development process (that is, certain tasks must be completed before others may start) and the resources that are available (for example, activity C follows activity B because Andy cannot work on both tasks at the same time). In drawing up the chart, we have therefore done two things – we have sequenced the tasks (that is, identified the dependencies among activities dictated by the development process) and scheduled them (that is, specified when they should take place). The

Separating the logical sequencing from the scheduling may be likened to the principle used in SSADM of separating the logical system from its physical implementation.

The bar chart does not show why certain decisions have been made. It is not clear, for example, why activity H is not scheduled to start until week 9. It could be that it cannot start until activity F has been completed or it might be because Charlie is going to be on holiday during week 8.

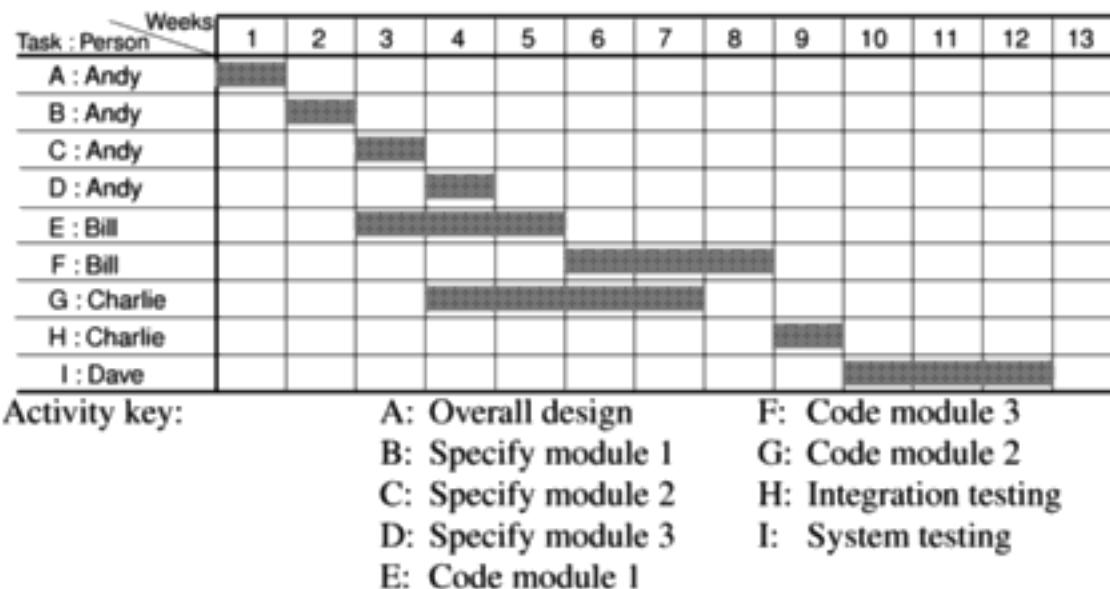


Figure 6.6 A project plan as a bar chart.

scheduling has had to take account of the availability of staff and the ways in which the activities have been allocated to them. The schedule might look quite different were there a different number of staff or were we to allocate the activities differently.

In the case of small projects, this combined sequencing–scheduling approach might be quite suitable, particularly where we wish to allocate individuals to particular tasks at an early planning stage. However, on larger projects it is better to separate out these two activities: to sequence the task according to their logical relationships and then to schedule them taking into account resources and other factors.

Approaches to scheduling that achieve this separation between the *logical* and the *physical* use networks to model the project and it is these approaches that we will consider in subsequent sections of this chapter.

6.7 Network planning models

CPM was developed by the Du Pont Chemical Company who published the method in 1958, claiming that it had saved them \$1 million in its first year of use.

These project scheduling techniques model the project's activities and their relationships as a network. In the network, time flows from left to right. These techniques were originally developed in the 1950s – the two best known being CPM (Critical Path Method) and PERT (Program Evaluation Review Technique). More recently a variation on these techniques, called *precedence networks*, has become popular and it is this method that is adopted in the majority of computer applications currently available. All three methods are very similar and it must be admitted that many people use the same name (particularly CPM) indiscriminately to refer to any or all of the methods.

In the following sections of this chapter, we will look at the critical path method and precedence networks – a discussion of PERT will be reserved for Chapter 7 when we look at risk analysis.

6.8 Formulating a network model

The first stage in creating a network model is to represent the activities and their interrelationships as a graph. In CPM we do this by representing activities as links (arrowed lines) in the graph – the nodes (circles) representing the events of activities starting and finishing.

In Chapter 2 we saw how Amanda used her Product Breakdown to obtain an activity network. Figure 6.7 shows the fragment of her network that was discussed in that chapter and Figure 6.8 shows how this network would look represented as a critical path network.

Case Study Example

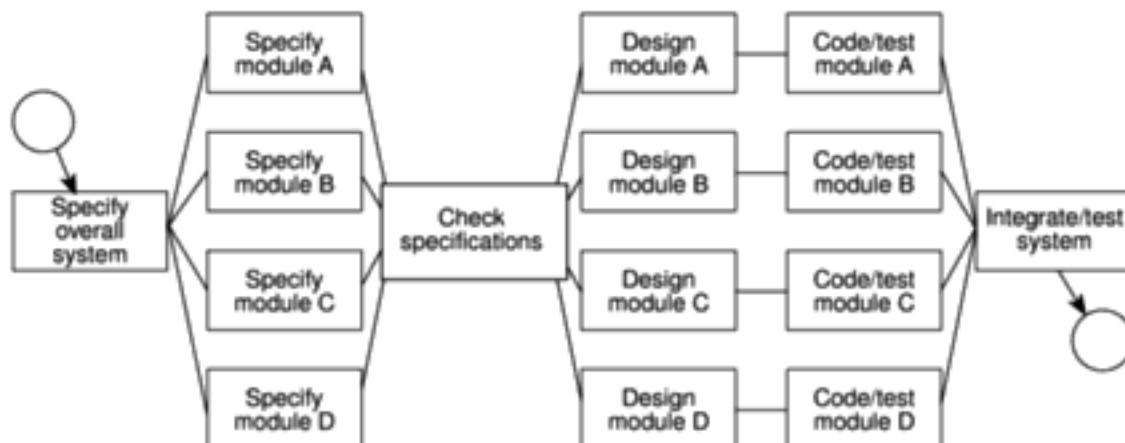


Figure 6.7 The IOE maintenance group accounts project activity network fragment with a checkpoint activity added.

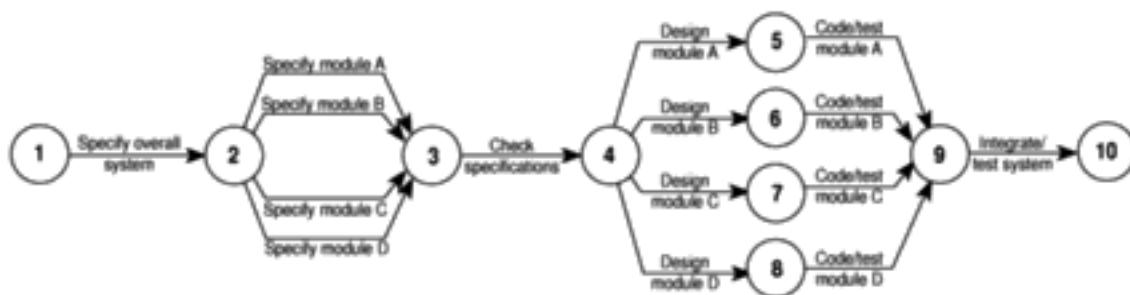


Figure 6.8 The IOE maintenance group accounts project activity network fragment represented as a CPM network.

Constructing CPM networks

Before we look at how CPM networks are used, it is worth spending a few moments considering the rules for their construction.

CPM networks are examples of directed graphs.

A project network may have only one start node The start node (node 1 in Figure 6.8) designates the point at which the project may start. All activities coming from that node may start immediately resources are available – that is, they do not have to wait for any other activities to be completed.

A project network may have only one end node The end node designates the completion of the project and a project may only finish once! The end node for the project fragment shown in Figure 6.8 is the one numbered 10.

A link has duration A link represents an activity and, in general, activities take time to execute. Notice, however, that the network in Figure 6.8 does not contain any reference to durations. The links are not drawn in any way to represent the activity durations. The network drawing merely represents the logic of the project – the rules governing the order in which activities are to be carried out.

Nodes have no duration Nodes are events and, as such, are instantaneous points in time. The *source node* is the event of the project becoming ready to start and the *sink node* is the event of the project becoming completed. Intermediate nodes represent two simultaneous events – the event of all activities leading in to a node having been completed and the event of all activities leading out of that node being in a position to be started.

In Figure 6.9 node 3 is the event that both coding and data take-on have been completed and activity program testing is free to start. Installation may be started only when event 4 has been achieved, that is, as soon as program testing has been completed.

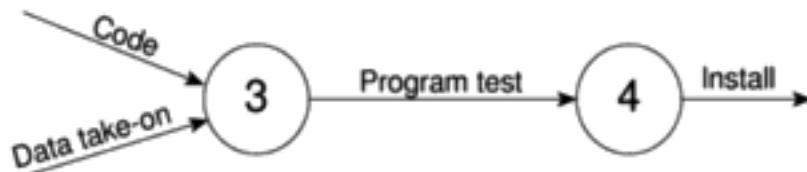


Figure 6.9 Fragment of a CPM network.

Time moves from left to right If at all possible, networks are drawn so that time moves from left to right. It is rare that this convention needs to be flouted but, in any case, the arrows on the activity lines give a strong visual indication of the time flow of the project.

Nodes are numbered sequentially There are no precise rules about node numbering but nodes should be numbered so that head nodes (those at the ‘arrow’ end of an activity) always have a higher number than tail events (those at the ‘non-arrow’ end of an activity). This convention makes it easy to spot loops.

A network may not contain loops Figure 6.10 demonstrates a loop in a CPM network. A loop is an error in that it represents a situation that cannot occur in practice. While loops, in the sense of iteration, may occur in practice, they cannot be directly represented in a project network. Note that the logic of Figure 6.10 suggests that program testing cannot start until the errors have been corrected.

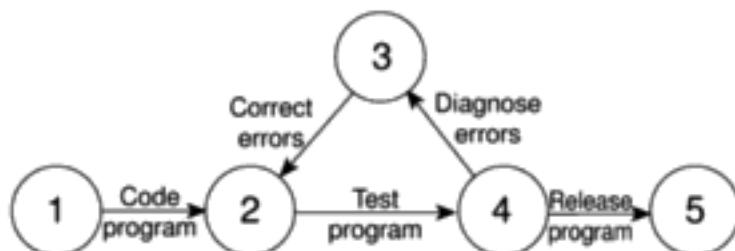


Figure 6.10 A loop represents an impossible sequence.

If we know the number of times we expect to repeat a set of activities, a test–diagnose–correct sequence, for example, then we can draw that set of activities as a straight sequence, repeating it the appropriate number of times. If we do not know how many times a sequence is going to be repeated then we cannot calculate the duration of the project unless we adopt an alternative strategy such as redefining the complete sequence as a single activity and estimating how long it will take to complete it.

A network may not contain dangles A dangling activity such as *Write user manual* in Figure 6.11 cannot exist, as it would suggest there are two completion points for the project. If, in Figure 6.11 node 5 represents the true project completion point and there are no activities dependent on activity *Write user manual*, then the network should be redrawn so that activity *Write user manual* starts at node 2 and terminates at node 5 – in practice, we would need to insert a dummy activity between nodes 3 and 5 as described in Section 6.9. In other words, all events, except the first and the last, must have at least one activity entering them and at least one activity leaving them and all activities must start and end with an event.

Precedents are the immediate preceding activities In Figure 6.9, the activity *Program test* cannot start until both *Code* and *Data take-on* have been completed and activity *Install* cannot start until *Program test* has finished. *Code* and *Data take-on* can therefore be said to be precedents of *Program test*, and *Program test* is a precedent of *Install*. Note that we do not speak of *Code* and *Data take-on* as precedents of *Install* – that relationship is implicit in the previous statement.

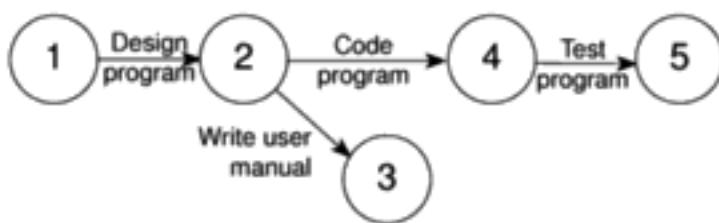


Figure 6.11 A dangle.

Dangles are not allowed in activity networks. Although undesirable, they are allowed in precedence networks (discussed in Chapter 9).

Exercise 6.1

Take a look at the networks in Figure 6.12. State what is wrong with each of them and where possible redraw them correctly.

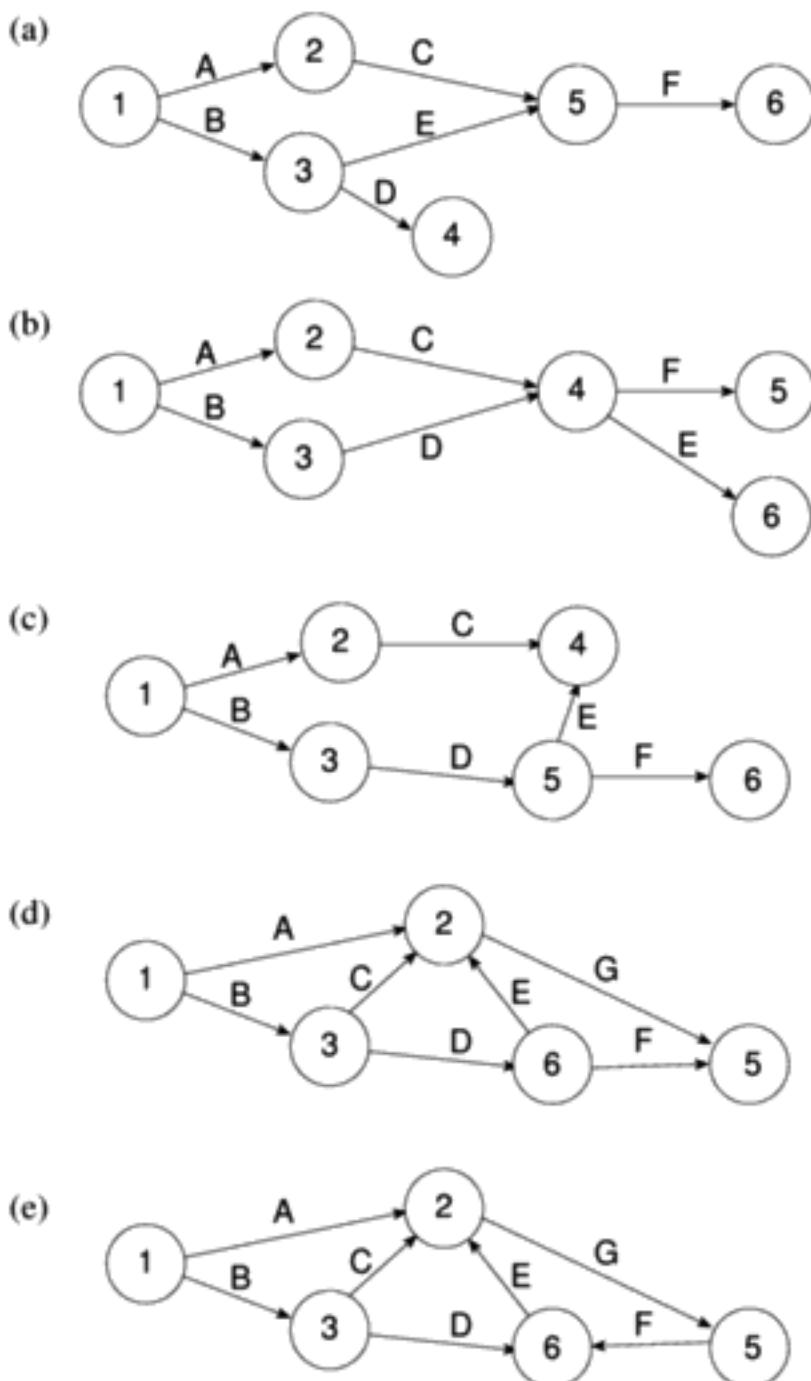


Figure 6.12 Some activity networks.

6.9 Using dummy activities

When two paths within a network have a common event although they are, in other respects independent, a logical error such as that illustrated in Figure 6.13 might occur.

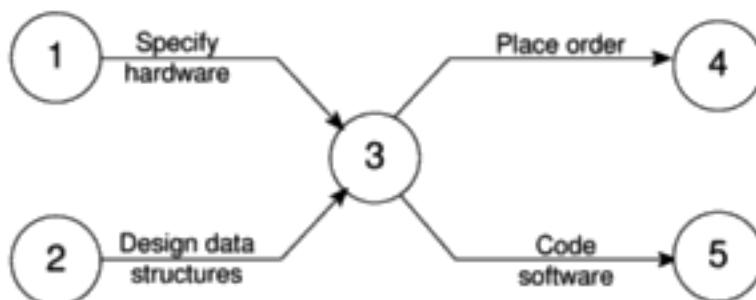


Figure 6.13 Two paths with a common node.

Suppose that, in a particular project, it is necessary to specify a certain piece of hardware before placing an order for it and before coding the software. Before coding the software it is also necessary to specify the appropriate data structures, although clearly we do not need to wait for this to be done before the hardware is ordered.

Figure 6.13 is an attempt to model the situation described above, although it is incorrect in that it requires both hardware specification and data structure design to be completed before either an order may be placed or software coding may commence.

We can resolve this problem by separating the two (more or less) independent paths and introducing a dummy activity to link the completion of data structure design to the start of the activity placing an order. This effectively breaks the link between data structure design and placing the order and is shown in Figure 6.14.

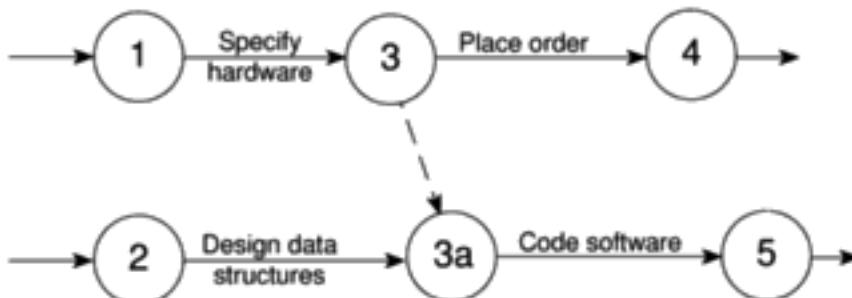


Figure 6.14 Two paths linked by a dummy activity.

Dummy activities, shown as dotted lines on the network diagram, have a zero duration and use no resources. They are often used to aid in the layout of network drawings as in Figure 6.15. The use of a dummy activity where two activities share the same start and end nodes makes it easier to distinguish the activity end-points.

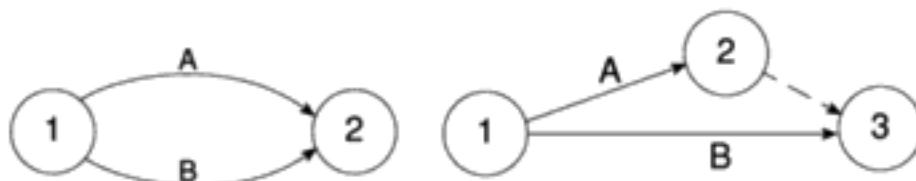


Figure 6.15 Another use of a dummy activity.

Exercise 6.2

Take another look at Brigette's college payroll activity network fragment, which you developed in Exercise 2.3 (or take a look at the model answer in Figure F.3). Redraw this as a CPM network.

6.10 Representing lagged activities

We might come across situations where we wished to undertake two activities in parallel so long as there is a lag between the two. We might wish to document amendments to a program as it was being tested – particularly if evaluating a prototype. In such a case we could designate an activity 'test and document amendments'. This would, however, make it impossible to show amendment recording starting after testing had begun and finishing a little after the completion of testing.

Where activities can occur in parallel with a time lag between them we represent these with pairs of dummy activities as shown in Figure 6.16. Where the activities are lagged because a stage in one activity must be completed before the other may proceed, it is likely to be better to show each stage as a separate activity.

Where parallel activities have a time lag we may show this as a 'ladder' of activities. In this case documentation may proceed alongside prototype testing so long as it starts at least a day later. It will finish two days after the completion of prototype testing.

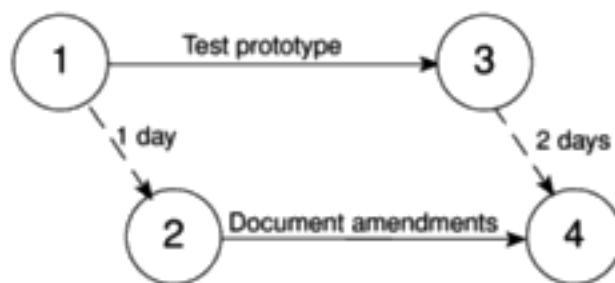


Figure 6.16 Using the ladder technique to indicate lags.

6.11 Adding the time dimension

Having created the logical network model indicating what needs to be done and the interrelationships between those activities, we are now ready to start thinking about when each activity should be undertaken.

The critical path method is concerned with two primary objectives: planning the project in such a way that it is completed as quickly as possible; and identifying

those activities where a delay in their execution is likely to affect the overall end date of the project or later activities' start dates.

The method requires that for each activity we have an estimate of its duration. The network is then analysed by carrying out a *forward pass*, to calculate the earliest dates at which activities may commence and the project be completed, and a *backward pass*, to calculate the latest start dates for activities and the *critical path*.

In practice we would use a software application to carry out these calculations for anything but the smallest of projects. It is important, though, that we understand how the calculations are carried out in order to interpret the results correctly and understand the limitations of the method.

The description and example that follow use the small example project outlined in Table 6.1 – a project composed of eight activities whose durations have been estimated as shown in the table.

Table 6.1 An example project specification with estimated activity durations and precedence requirements

Activity	Duration (weeks)	Precedents
A Hardware selection	6	
B Software design	4	
C Install hardware	3	A
D Code & test software	4	B
E File take-on	3	B
F Write user manuals	10	
G User training	3	E, F
H Install & test system	2	C, D

CPM conventions

There are a number of differing conventions that have been adopted for entering information on a CPM network. Typically the diagram is used to record information about the events rather than the activities – activity-based information (other than labels or descriptions) is generally held on a separate activity table.

One of the more common conventions for labelling nodes, and the one adopted here, is to divide the node circle into quadrants and use those quadrants to show the event number, the latest and earliest dates by which the event should occur, and the event slack (which will be explained later).



Draw an activity network using CPM conventions for the project specified in Table 6.1. When you have completed it, compare your result with that shown in Figure 6.17.

Exercise 6.3

Figure 6.17 illustrates the network for the project specified in Table 6.1.

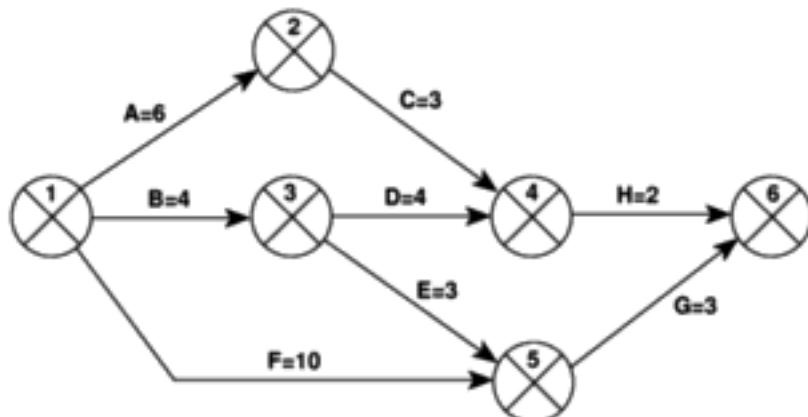


Figure 6.17 The CPM network for the example project.

6.12 The forward pass

The forward pass is carried out to calculate the earliest date on which each event may be achieved and the earliest date on which each activity may be started and completed. The earliest date for an event is the earliest date by which all activities upon which it depends can be completed.

By convention, dates indicate the end of the period and the project is therefore shown as starting in week zero (or the beginning of week 1).

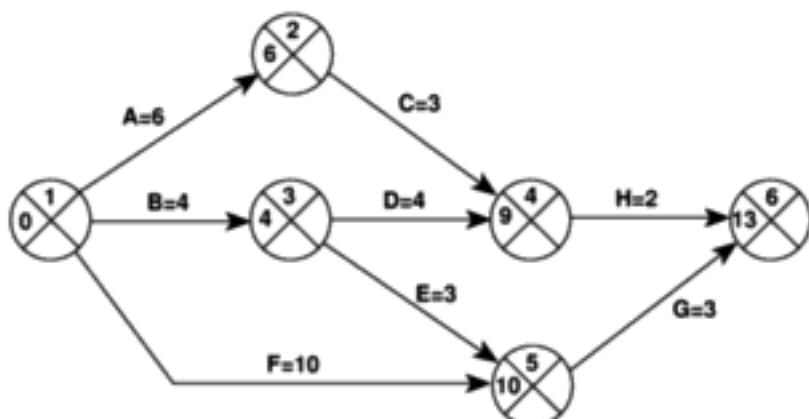
The forward pass and the calculation of earliest start dates is calculated according to the following reasoning.

- Activities A, B and F may start immediately, so the earliest date for event 1 is zero and the earliest start date for these three activities is also zero.
- Activity A will take 6 weeks, so the earliest it can finish is week 6 (recorded in the activity table). Therefore the earliest we can achieve event 2 is week 6.
- Activity B will take 4 weeks, so the earliest it can finish and the earliest we can achieve event 3 is week 4.
- Activity F will take 10 weeks, so the earliest it can finish is week 10 – we cannot, however, tell whether or not this is also the earliest date that we can achieve event 5 since we have not, as yet, calculated when activity E will finish.
- Activity E can start as early as week 4 (the earliest date for event 3) and, since it is forecasted to take 3 weeks, will be completed, at the earliest, at the end of week 7.
- Event 5 may be achieved when both E and F have been completed, that is, week 10 (the later of 7 and 10).
- Similarly we can reason that event 4 will have an earliest date of week 9. This is the later of the earliest finish for activity D (week 8) and the earliest finish for activity C (week 9).

During the forward pass, earliest dates are recorded as they are calculated. For events, they are recorded on the network diagram and for activities they are recorded on the activity table.

- The earliest date for the completion of the project, event 6, is therefore the end of week 13 – the later of 11 (the earliest finish for H) and 13 (the earliest finish for G).

The results of the forward pass are shown in Figure 6.18 and Table 6.2.



The forward pass rule:
the earliest date for an event is the earliest finish date for all the activities terminating at that event. Where more than one activity terminates at a common event we take the latest of the earliest finish dates for those activities.

Figure 6.18 A CPM network after the forward pass.

Table 6.2 The activity table after the forward pass

Activity	Duration (weeks)	Earliest start date	Latest start date	Earliest finish date	Latest finish date	Total float
A	6	0		6		
B	4	0		4		
C	3	6		9		
D	4	4		8		
E	3	4		7		
F	10	0		10		
G	3	10		13		
H	2	9		11		

6.13 The backward pass

The second stage is to carry out a backward pass to calculate the latest date at which each event may be achieved, and each activity started and finished, without delaying the end date of the project. The latest date for an event is the latest date by which all immediately following activities must be started for the project to be completed on time. In calculating the latest dates, we assume that the latest finish date for the project is the same as the earliest finish date – that is, we wish to complete the project as early as possible.

Figure 6.19 illustrates our network and Table 6.3 the activity table after carrying out the backward pass – as with the forward pass, event dates are recorded on the diagram and activity dates on the activity table.

The backward pass rule: the latest date for an event is the latest start date for all the activities that may commence from that event. Where more than one activity commences at a common event we take the earliest of the latest start dates for those activities.

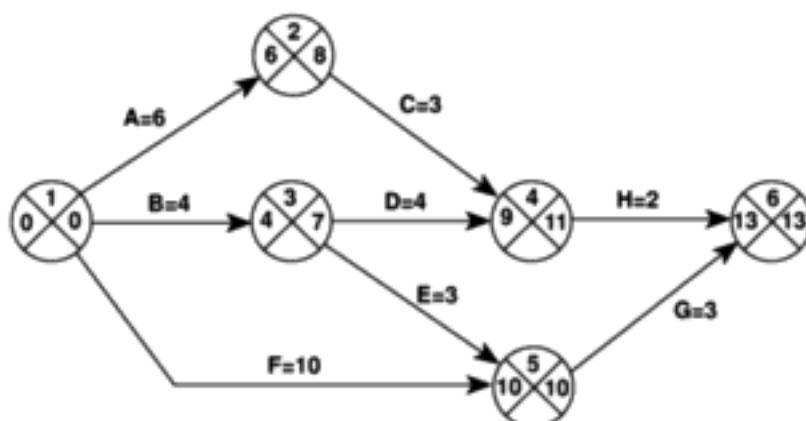


Figure 6.19 The CPM network after the backward pass.

Table 6.3 The activity table following the backward pass

Activity	Duration (weeks)	Earliest start date	Latest start date	Earliest finish date	Latest finish date	Total float
A	6	0	2	6	8	
B	4	0	3	4	7	
C	3	6	8	9	11	
D	4	4	7	8	11	
E	3	4	7	7	10	
F	10	0	0	10	10	
G	3	10	10	13	13	
H	2	9	11	11	13	

The latest event dates are calculated as follows.

- The latest date for node 6 is assumed to be week 13, the same as the earliest date.
- The latest date for event 5 is week 10, since activity G will take 3 weeks and must be completed by week 13 if the project end date is not to be exceeded.
- The latest date for event 4 is week 11 since activity H does not need to be started until week 11 if it takes 2 weeks and does not need to be completed until week 13.
- The latest date for event 3 is the latest date by which we must be in a position to start both activities D and E. Activity E need not finish until week 10 and

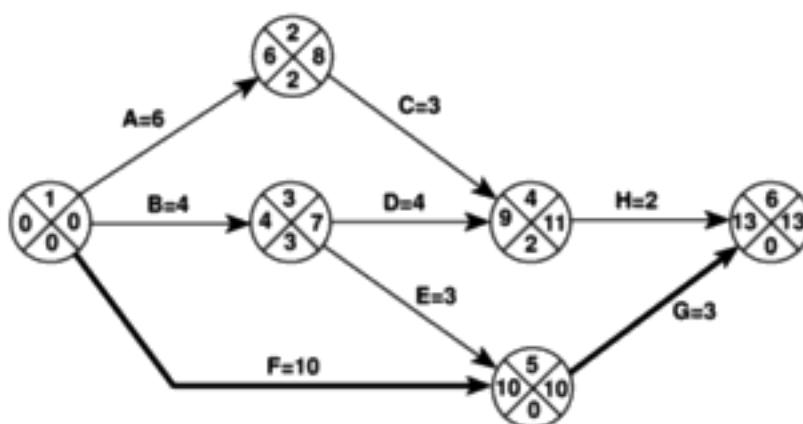
need not therefore start until week 7. Activity D need not finish until week 11 and, having a duration of 4 weeks, need not start until week 7. The latest date for event 3 is therefore week 7.

- The latest date for event 2 is week 8 since C, which takes 3 weeks, need not be finished until week 11.
- The earliest and latest dates for the start event must always be the same unless an arithmetic error has occurred.
- The latest date for event 1 is the latest by which we must be in a position to start activity A (which must start by week 2), activity B (which must start by week 3) and activity F (which must start by week 0). This event's latest date is therefore zero. This is, of course, not very surprising since it tells us that if the project does not start on time it won't finish on time.

6.14 Identifying the critical path

Any delay on the critical path will delay the project.

The difference between the earliest date and the latest date for an event is known as the *slack* – it is a measure of how late an event may be without affecting the end date of the project. Any event with a slack of zero is critical in the sense that any delay in achieving that event will delay the completion date of the project as a whole. There will always be at least one path through the network joining those critical events – this path is known as the *critical path* (Figure 6.20).



The critical path is the longest path through the network.

Figure 6.20 The critical path.

The significance of the critical path is two-fold.

- In managing the project, we must pay particular attention to monitoring activities on the critical path so that the effects of any delay or resource unavailability are detected and corrected at the earliest opportunity.
- In planning the project, it is the critical path that we must shorten if we are to reduce the overall duration of the project.

6.15 Activity float

Whereas events have slack, activities possess *float*. The total float shown in Table 6.4 is the difference between the earliest start date of an activity and its latest start (or the difference between its earliest finish and its latest finish). It tells us by how long the activity's start or completion may be delayed without affecting the end date of the project.

Table 6.4 *The activity schedule showing total float for each activity*

Activity	Duration (weeks)	Earliest start date	Latest start date	Earliest finish date	Latest finish date	Total float
A	6	0	2	6	8	2
B	4	0	3	4	7	3
C	3	6	8	9	11	2
D	4	4	7	8	11	3
E	3	4	7	7	10	3
F	10	0	0	10	10	0
G	3	10	10	13	13	0
H	2	9	11	11	13	2

Total float may only be used once.

Although the total float is shown for each activity, it really 'belongs' to a path through the network. Activities A and C each have 2 weeks total float. If, however, activity A uses up its float (that is, it is not completed until week 8) then activity B will have zero float (it will have become critical). In such circumstances it may be misleading and detrimental to the project's success to publicize total float!

There are a number of other measures of activity float, including the following.

- **Free float:** the time by which an activity may be delayed without affecting any subsequent activity. It is calculated as the difference between the earliest completion date for the activity and the earliest start date of the succeeding activity. This might be considered a more satisfactory measure of float for publicizing to the staff involved in undertaking the activities.
- **Interfering float:** the difference between total float and free float. This is quite commonly used, particularly in association with the free float. Once the free float has been used (or if it is zero), the interfering float tells us by how much the activity may be delayed without delaying the project end date – even though it will delay the start of subsequent activities.

Exercise 6.4

Calculate the free float and interfering float for each of the activities shown in the activity network (Table 6.4).

6.16 Shortening the project duration

If we wish to shorten the overall duration of a project we would normally consider attempting to reduce activity durations. In many cases this can be done by applying more resources to the task – working overtime or procuring additional staff, for example. The critical path indicates where we must look to save time – if we are trying to bring forward the end date of the project, there is clearly no point in attempting to shorten non-critical activities. Referring to Figure 6.20 it can be seen that we could complete the project in week 12 by reducing the duration of activity F by one week (to 9 weeks).

Referring to Figure 6.20, suppose that the duration for activity F is shortened to 8 weeks. Calculate the end date for the project.

What would the end date for the project be if activity F were shortened to 7 weeks? Why?

Exercise 6.5

As we reduce activity times along the critical path we must continually check for any new critical path emerging and redirect our attention where necessary.

There will come a point when we can no longer safely, or cost-effectively, reduce critical activity durations in an attempt to bring forward the project end date. Further savings, if needed, must be sought in a consideration of our work methods and by questioning the logical sequencing of activities. Generally, time savings are to be found by increasing the amount of parallelism in the network and the removal of bottlenecks (subject always, of course, to resource and quality constraints).

6.17 Identifying critical activities

The critical path identifies those activities which are critical to the end date of the project; however, activities that are not on the critical path may become critical. As the project proceeds, activities will invariably use up some of their float and this will require a periodic recalculation of the network. As soon as the activities along a particular path use up their total float then that path will become a critical path and a number of hitherto non-critical activities will suddenly become critical.

It is therefore common practice to identify ‘near-critical’ paths – those whose lengths are within, say, 10–20% of the duration of the critical path or those with a total float of less than, say, 10% of the project’s uncompleted duration.

The importance of identifying critical and near-critical activities is that it is they that are most likely to be the cause of delays in completing the project. We shall see, in the next three chapters, that identifying these activities is an important step in risk analysis, resource allocation and project monitoring.

For a more in-depth discussion of the role of the critical path in project monitoring, see Chapter 9.

6.18 Precedence networks

Activity label	Duration	
Earliest start	Activity description	Earliest finish
Latest start		Latest finish
Activity span	Float	

Activity boxes allow substantial detail to be recorded on a precedence network.

Where CPM networks use links to represent activities and nodes to represent events, precedence networks use boxes (nodes) to represent activities (sometimes known as work items) and links to represent dependencies. The boxes may carry task descriptions and duration estimates and the links may contain a duration denoting a lag between the completion of one task and the start of the next.

In the notation we use here, the items indicated in the boxes are the same as those shown on the CPM network and activity table with the addition of *activity span*. Activity span is the difference between the earliest start date and the latest finish date and is a measure of the maximum time allowable for the activity.

Proponents of precedence networks claim that they are easier to draw neatly, dummy activities are virtually redundant and it is easier for people to interpret them. Figure 6.21 shows the example project in Figure 6.20 drawn as a precedence network. As you can see, it contains much more information than the CPM network and we do not need to keep a separate activity table. The critical path through activities F and G is shown as a heavy line.

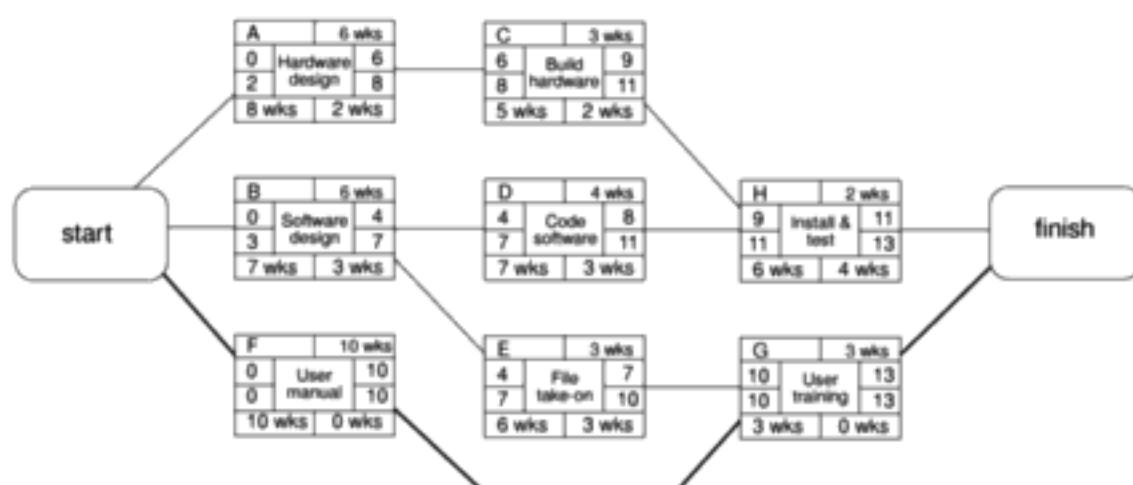


Figure 6.21 A precedence network.

A further advantage of precedence networks is that they can represent parallel lagged activities. The example shown in Figure 6.16, which required the use of dummy activities in a CPM network, may be represented much more elegantly in a precedence network as shown in Figure 6.22.

Documenting amendments may take place alongside prototype testing so long as it starts at least one day later and finishes two days later.

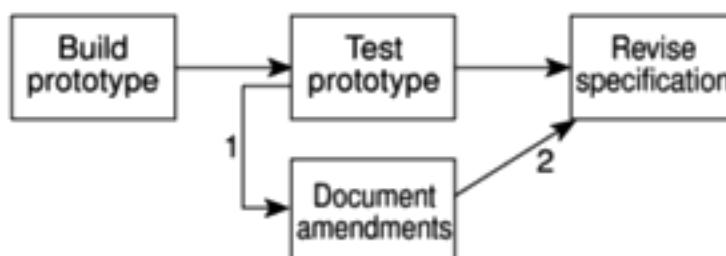


Figure 6.22 Parallel lagged activities in a precedence network.

Analysis of precedence networks proceeds in exactly the same ways as discussed above – we carry out a forward and backward pass to calculate earliest and latest start and finish dates and identify the critical path.

Refer back to Amanda's CPM network illustrated in Figure 6.8 and redraw it as a precedence network.

Exercise 6.6

Using the activity durations given in Table 6.5, calculate the earliest completion date for the project and identify the critical path on your network.

Table 6.5 Estimated activity durations for Amanda's network

Activity	Estimated duration (days)	Activity	Estimated duration (days)
Specify overall	34	Design module C	4
Specify module A	20	Design module D	4
Specify module B	15	Code/test module A	30
Specify module C	25	Code/test module B	28
Specify module D	15	Code/test module C	15
Check specification	2	Code/test module D	25
Design module A	7	System integration	6
Design module B	6		

6.19 Conclusion

In this chapter, we have discussed the use of the critical path method and precedence networks to obtain an ideal activity plan. This plan tells us the order in which we should execute activities and the earliest and latest we can start and finish them.

These techniques help us to identify which activities are critical to meeting a target completion date.

In order to manage the project we need to turn the activity plan into a schedule that will specify precisely when each activity is scheduled to start and finish. Before we can do this, we must consider what resources will be required and whether or not they will be available at appropriate times. As we shall see, the allocation of resources to an activity may be affected by how we view the importance of the task and the risks associated with it. In the next two chapters we look at these aspects of project planning before we consider how we might publish a schedule for the project.

6.20 Further exercises

1. Draw an activity network using either CPM or precedence network conventions for each of the following projects:
 - getting married;
 - choosing and purchasing a desktop computer;
 - organizing and carrying out a survey of users' opinions of an information system.
2. If you have access to a project planning application, use it to produce a project plan for the IOE maintenance group accounts project. Base your plan on that used for Exercise 6.6 and verify that your application reports the same information as you calculated manually when you did the exercise.
3. Based on your answer to Exercise 6.6, discuss what options Amanda might consider if she found it necessary to complete the project earlier than day 104?

Chapter 7

Risk management

OBJECTIVES

After completing this chapter you will be able to:

- identify the factors putting a project at risk;
 - categorize and prioritize action for risk elimination or containment;
 - quantify the likely effects of risk on project time-scales.
-

7.1 Introduction

In Chapter 3 we considered project evaluation, including assessment of the risk of the project's not delivering the expected benefits. In this chapter, we are concerned with the risk of the development project's not proceeding according to plan. We are primarily concerned with the risks of the project's running late or over budget and with the identification of the steps that can be taken to avoid or minimize those risks.

Some risks are more important than others. Whether or not a particular risk is important depends on the nature of the risk, its likely effects on a particular activity and the criticality of the activity. High risk activities on a project's critical path are a cause for concern.

To reduce these dangers, we must ensure that risks are minimized or, at least, distributed over the project and, ideally, removed from critical path activities.

The risk of an activity's running over time is likely to depend, at least in part, on who is doing or managing it. The evaluation of risk and the allocation of staff and other resources are therefore closely connected.

The allocation of resources is discussed in the next chapter.

7.2 The nature of risk

For the purpose of identifying and managing those risks that may cause a project to overrun its time-scale or budget, it is convenient to identify three types of risk:

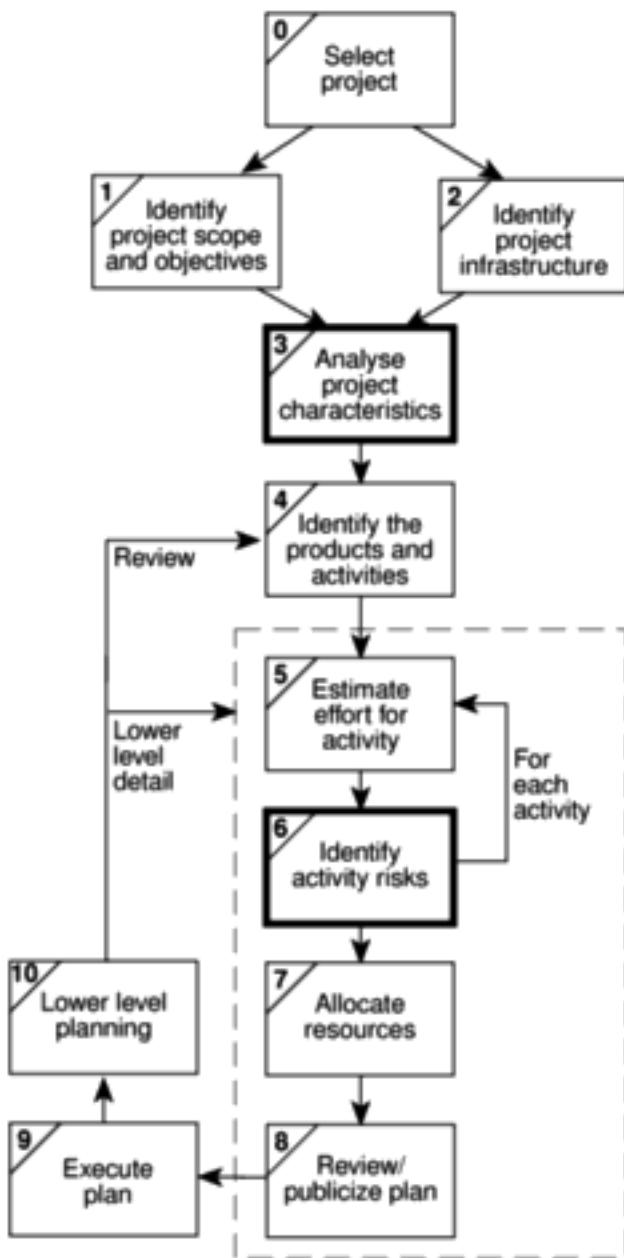


Figure 7.1 Risk analysis is carried out in Steps 3 and 6.

- those caused by the inherent difficulties of estimation;
- those due to assumptions made during the planning process;
- those of unforeseen (or at least unplanned) events occurring.

Estimation errors

Improved quality control should make it easier to predict the time required for program and system testing.

Some tasks are harder to estimate than others because of the lack of experience of similar tasks or because of the nature of a task. Producing a set of user manuals is reasonably straightforward and, given that we have carried out similar tasks previously, we should be able to estimate with some degree of accuracy how long it will take and how much it will cost. On the other hand, the time required for

program testing and debugging, might be difficult to predict with a similar degree of accuracy – even if we have written similar programs in the past.

Estimation can be improved by analysing historic data for similar activities and similar systems. Keeping records comparing our original estimates with the final outcome will reveal the type of tasks that are difficult to estimate correctly.

See Chapter 5 for methods of estimation.

Planning assumptions

At every stage during planning, assumptions are made which, if not valid, may put the plan at risk. Our activity network, for example, is likely to be built on the assumption of using a particular design methodology – which may be subsequently changed. We generally assume that, following coding, a module will be tested and then integrated with others – we might not plan for module testing showing up the need for changes in the original design but, in the event, it might happen.

At each stage in the planning process, it is important to list explicitly all of the assumptions that have been made and identify what effects they might have on the plan if they are inappropriate.

Eventualities

Some eventualities might never be foreseen and we can only resign ourselves to the fact that unimaginable things do, sometimes, happen. They are, however, very rare. The majority of unexpected events can, in fact, be identified – the requirements specification might be altered after some of the modules have been coded, the senior programmer might take maternity leave, the required hardware might not be delivered on time. Such events do happen from time to time and, although the likelihood of any one of them happening during a particular project may be relatively low, they must be considered and planned for.

7.3 Managing risk

The objective of risk management is to avoid or minimize the adverse effects of unforeseen events by avoiding the risks or drawing up contingency plans for dealing with them.

There are a number of models for risk management, but most are similar, in that they identify two main components – *risk identification* and *risk management*. An example of an often-used model is that in Figure 7.2, which shows a task breakdown structure for what Barry Boehm calls *risk engineering*.

- **Risk identification** consists of listing all of the risks that can adversely affect the successful execution of the project. We discuss this in more detail in Section 7.4.
- **Risk estimation** consists of assessing the likelihood and impact of each hazard. We discuss this in more detail in Section 7.5 under the broader topic of risk analysis.

This is based on the breakdown presented by Barry Boehm in his *Tutorial on Software Risk Management*, IEEE Computer Society, 1989.



Figure 7.2 Boehm's risk engineering task breakdown.

- **Risk evaluation** consists of ranking the risks and determining risk aversion strategies. We discuss this in Section 7.5 on risk analysis and Section 7.6 where we discuss strategies for risk aversion.
- **Risk planning** consists of drawing up contingency plans and, where appropriate, adding these to the project's task structure. With small projects, risk planning is likely to be the responsibility of the project manager but medium or large projects will benefit from the appointment of a full-time risk manager.
- **Risk control** concerns the main functions of the risk manager in minimising and reacting to problems throughout the project. This function will include aspects of quality control in addition to dealing with problems as they occur.
- **Risk monitoring** must be an ongoing activity, as the importance and likelihood of particular risks can change as the project proceeds. Risk monitoring is discussed in Chapter 9.
- **Risk directing** and **risk staffing** are concerned with the day-to-day management of risk. Risk aversion and problem solving strategies frequently involve the use of additional staff and this must be planned for and directed.

Whatever task model or whichever techniques are used, risk management will not be effective unless all project staff are risk-oriented and are provided with an environment where they can freely discuss the risks that might affect a project. All too often, team members who identify potential risks at an early stage are seen as having a negative attitude.

Writing about attitudes to risk, Dwayne Phillips remarks that 'I have seen a room get suddenly quiet when someone brings up a "concern"' but says that 'pretending that problems will not occur will not prevent them'. For effective risk management, it is important that the project team are encouraged to identify and discuss risks as early as possible in the project's life.

The techniques described in the rest of this chapter describe how risks can be identified and quantified and are designed to provide a framework that engenders a positive attitude to the analysis and management of project risks.

7.4 Risk identification

The first stage in any risk assessment exercise is to identify the hazards that might affect the duration or resource costs of the project. A hazard is an event that might occur and will, if it does occur, create a problem for the successful completion of the project. In identifying and analysing risks, we can usefully distinguish between the cause (or hazard), its immediate effect (the problem that it creates) and the risk that it will pose to the project.

For example, the illness of a team member is a *hazard* that might result in the *problem* of late delivery of a component. The late delivery of that component is likely have an effect on other activities and might, particularly if it is on the critical path, put the project completion date at *risk*.

A common way of identifying hazards is to use a checklist listing all the possible hazards and factors that influence them. Typical checklists list many, even hundreds, of factors and there are, today, a number of knowledge-based software products available to assist in this analysis.

Some hazards are *generic risks* – that is, they are relevant to all software projects and standard checklists can be used and augmented from an analysis of past projects to identify them. These will include risks such as misunderstanding the requirements or key personnel being ill. There will also be *specific risks* that are relevant to an individual project and these are likely to be more difficult to identify without an involvement of the members of the project team and a working environment that encourages risk assessment.

The categories of factors that will need to be considered include the following.

Application factors The nature of the application – whether it is a simple data processing application, a safety-critical system or a large distributed system with real-time elements – is likely to be a critical factor. The expected size of the application is also important – the larger the system, the greater is the likelihood of errors and communication and management problems.

Some of these issues have been addressed in Chapter 4.

Staff factors The experience and skills of the staff involved are clearly major factors – an experienced programmer is, one would hope, less likely to make errors than one with little experience. We must, however, also consider the appropriateness of the experience – experience in coding small data processing modules in Cobol may be of little value if we are developing a complex real-time control system using C++.

The effect of staff experience was considered in Chapter 5 on effort estimation.

Such factors as the level of staff satisfaction and the staff turn-over rates are also important to the success of any project – demotivated staff or key personnel leaving unexpectedly have caused many a project to fail.

Project factors It is important that the project and its objectives are well defined and that they are absolutely clear to all members of the project team and all key stakeholders. Any possibility that this is not the case will pose a risk to the success of the project. Similarly, an agreed and formal quality plan must be in place and adhered to by all participants and any possibility that the quality plan is inadequate or not adhered to will jeopardize the project.

Project methods Using well specified and structured methods (such as PRINCE 2 and SSADM) for project management and system development will decrease the risk of delivering a system that is unsatisfactory or late. Using such methods for the first time, though, may cause problems and delays – it is only with experience that the benefits accrue.

Hardware/software factors A project that requires new hardware for development is likely to pose a higher risk than one where the software can be developed on existing (and familiar) hardware. Where a system is developed on one type of hardware or software platform to be used on another there might be additional (and high) risks at installation.

Changeover factors The need for an 'all-in-one' changeover to the new system poses particular risks. Incremental or gradual changeover minimizes the risks involved but is not always practical. Parallel running can provide a safety net but might be impossible or too costly.

Supplier factors The extent to which a project relies on external organizations that cannot be directly controlled often influences the project's success. Delays in, for example, the installation of telephone lines or delivery of equipment may be difficult to avoid – particularly if the project is of little consequence to the external supplier.

Environment factors Changes in the environment can affect a project's success. A significant change in the taxation regulations could, for example, have serious consequences for the development of a payroll application.

Health and safety factors While not generally a major issue for software projects (compared, say, to civil engineering projects), the possible effects of project activities on the health and safety of the participants and the environment should be considered. BS 6079 states that 'every project should include an audit of these specific risks before work starts' and that 'audit updates should be scheduled as part of the overall project plan'.

Exercise 7.1

Brigette finds that Brightmouth HE College does not have a project hazard checklist or questionnaire and decides to produce her own for the payroll project. List at least one question that she might include under each of the above headings.

Although some factors might influence the project as a whole, it is necessary to consider them individually for each activity – a key member of staff being ill

during fact-finding might, for example, be far less serious than a similar absence during user training. Within a PRINCE 2 environment it can be appropriate to list the factors for each of the products identified in the product breakdown structure.

7.5 Risk analysis

Having identified the risks that might affect our project we need some way of assessing their importance. Some risks will be relatively unimportant (for example, the risk that some of the documentation is delivered a day late), whereas some will be of major significance (such as the risk that the software is delivered late). Some are quite likely to occur (it is quite likely, for example, that one of the software developers in a team will take a few days sick leave during a lengthy project), whereas others are relatively unlikely (hardware failure causing loss of completed code, perhaps).

The probability of a hazard's occurring is known as the *risk likelihood*; the effect that the resulting problem will have on the project, if it occurs, is known as the *risk impact* and the importance of the risk is known as the *risk value* or *risk exposure*. The risk value is calculated as:

$$\text{risk exposure} = \text{risk likelihood} \times \text{risk impact}$$

Ideally the risk impact is estimated in monetary terms and the likelihood assessed as a probability. In that case the risk exposure will represent an expected cost in the same sense that we calculated expected costs and benefits when discussing cost–benefit analysis. The risk exposures for various risks can then be compared with each other to assess the relative importance of each risk and they can be directly compared with the costs and likelihoods of success of various contingency plans.

However, estimation of these costs and probabilities is likely to be difficult, subjective, time-consuming and costly. In spite of this, it is valuable to obtain some quantitative measure of risk likelihood and impact because, without these, it is difficult to compare or rank risks in a meaningful way. Moreover, the effort put into obtaining a good quantitative estimate can provide a deeper and valuable understanding of the problem.

Many risk managers use a simple scoring method to provide a quantitative measure for assessing each risk. Some just categorize likelihoods and impacts as high, medium or low, but this form of ranking does not allow the calculation of a risk exposure. A better and popular approach is to score the likelihood and impact on a scale of, say, 1 to 10 where the hazard that is most likely to occur receives a score of 10 and the least likely a score of 1.

Ranking likelihoods and impacts on a scale of 1 to 10 is relatively easy, but most risk managers will attempt to assign scores in a more meaningful way such that, for example, a likelihood scoring 8 is considered twice as likely as one with a score of 4.

These terms are common but not universal – you might come across alternative terms.

Expected costs and benefits were used in cost–benefit analysis in Chapter 3.

Impact measures, scored on a similar scale, must take into account the total risk to the project. This must include the following potential costs:

- the cost of delays to scheduled dates for deliverables;
- cost overruns caused by using additional or more expensive resources;
- the costs incurred or implicit in any compromise to the system's quality or functionality.

Table 7.1 illustrates part of Amanda's risk value assessment. Notice that the hazard with the highest risk value might not be the one that is most likely nor the one with the greatest potential impact.

Table 7.1 Part of Amanda's risk exposure assessment

	Hazard	Likelihood	Impact	Risk exposure
R1	Changes to requirements specification during coding	1	8	8
R2	Specification takes longer than expected	3	7	21
R3	Staff sickness affecting critical path activities	5	7	35
R4	Staff sickness affecting non-critical activities	10	3	30
R5	Module coding takes longer than expected	4	5	20
R6	Module testing demonstrates errors or deficiencies in design	1	10	10

Prioritizing the risks

Managing risk involves the use of two strategies:

- reducing the risk exposure by reducing the likelihood or impact;
- drawing up contingency plans to deal with the risk should it occur.

Any attempt to reduce a risk exposure or put a contingency plan in place will have a cost associated with it. It is therefore important to ensure that this effort is applied in the most effective way and we need a way of prioritizing the risks so that the more important ones can receive the greatest attention.

Exercise 7.2

Consider Amanda's risk exposure analysis shown in Table 7.1 and add some of the hazards that you considered when answering Exercise 7.1.

Estimate values for the likelihood and impact of each of these items and calculate their risk exposures.

Rank each of your risks according to their risk exposure and, assuming that Amanda does not have the time or resources to deal with all of them, try to categorize each of them as high, medium or low priority.

Risk exposures based on scoring methods must be treated with some caution. Amanda's assessment shown in Table 7.1 does not indicate, for example, that risk R5 is twice as important as R6. Nor can it be taken as necessarily meaning that R2 is more important than R5. In the first case, this is because we cannot interpret the risk exposure values quantitatively because they are based on a non-cardinal scoring method. In the second case, the exposure values are far too close for us to be able to distinguish between them – particularly in view of the somewhat approximate and subjective way in which Amanda is likely to have assessed the likelihoods and, perhaps to a lesser extent, the impacts.

The risk exposures will, however, allow us to obtain an approximate ranking in order of importance. Considering just the risks in Table 7.1, R3 and R4 are, on this basis, clearly the most important and we could classify them as being high risk concerns. There is a significant difference between the exposure scores of these two and one with the next highest exposure, R2. R2 and R5 have similar scores and might be thought of as medium priority risks. The two remaining risks, R1 and R6 have quite low exposure values and can therefore be classified as low risk items.

In practice, there are generally other factors, in addition to the risk exposure value, that must also be taken into account when prioritizing risks.

- **Confidence of the risk assessment** Some of our risk exposure assessments will be relatively poor. Where this is the case, there is a need for further investigation before action can be planned.
- **Compound risks** Some risks will be dependant on others. Where this is the case, they should be treated together as a single risk.
- **The number of risks** There is a limit to the number of risks that can be effectively considered and acted on by a project manager. We might therefore wish to limit the size of the prioritized list.
- **Cost of action** Some risks, once recognized, can be reduced or avoided immediately with very little cost or effort and it is sensible to take action on these regardless of their risk value. For other risks we need to compare the costs of taking action with the benefits of reducing the risk. One method for doing this is to calculate the *risk reduction leverage* (RRL) using the equation

$$\text{RRL} = \frac{\text{RE}_{\text{before}} - \text{RE}_{\text{after}}}{\text{risk reduction cost}}$$

Classifying risks into these three categories is clearly not always as easy as in this example although, in practice, risks do frequently cluster and break points are often quite distinct.

The RRL is used as a factor in prioritizing risks and for evaluating alternative courses of action in dealing with a particular risk.

where RE_{before} is the original risk exposure value, RE_{after} is the expected risk exposure value after taking action and the risk reduction cost is the cost of implementing the risk reduction action. Risk reduction costs must be expressed in the same units as risk values – that is, expected monetary values or score values. If the values are expected monetary values then an RRL greater than one indicates that we can expect to gain from implementing the risk reduction plan because the expected reduction in risk exposure is greater than the cost of the plan. In either case the higher the leverage value for a risk then the more worthwhile it will be to plan the risk reduction action.

7.6 Reducing the risks

Broadly, there are five strategies for risk reduction.

- **Hazard prevention** Some hazards can be prevented from occurring or their likelihood reduced to insignificant levels. The risk of key staff being unavailable for meetings can be minimized by early scheduling, for example.
- **Likelihood reduction** Some risks, while they cannot be prevented, can have their likelihoods reduced by prior planning. The risk of late changes to a requirements specification can, for example, be reduced by prototyping. Prototyping will not eliminate the risk of late changes and will need to be supplemented by contingency planning.
- **Risk avoidance** A project can, for example, be protected from the risk of overrunning the schedule by increasing duration estimates or reducing functionality.
- **Risk transfer** The impact of some risks can be transferred away from the project by, for example, contracting out or taking out insurance.
- **Contingency planning** Some risks are not preventable and contingency plans will need to be drawn up to reduce the impact should the hazard occur. A project manager should draw up contingency plans for using agency programmers to minimize the impact of any unplanned absence of programming staff.

In Section 7.4 we mentioned the use of checklists for hazard identification. Many of these generic checklists, as well as listing common generic hazards, list typical actions for risk reduction. The checklist in Table 7.2 is based upon an often-quoted list produced by Barry Boehm.

Exercise 7.3

For each of the risks listed in Table 7.1, identify actions that Amanda might take to reduce their likelihood or impact.

Table 7.2 Software projects risks and strategies for risk reduction.

<i>Risk</i>	<i>Risk reduction techniques</i>
Personnel shortfalls	staffing with top talent; job matching; team building; training and career development; early scheduling of key personnel.
Unrealistic time and cost estimates	multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods.
Developing the wrong software functions	improved project evaluation; formal specification methods; user surveys; prototyping; early users' manuals.
Developing the wrong user interface	prototyping; task analysis; user involvement.
Gold plating	requirements scrubbing; prototyping; cost–benefit analysis; design to cost.
Late changes to requirements	stringent change control procedures; high change threshold; incremental prototyping; incremental development (defer changes).
Shortfalls in external supplied components	benchmarking; inspections; formal specifications; contractual agreements; quality assurance procedures and certification.
Shortfalls in externally performed tasks	quality assurance procedures; competitive design or prototyping; teambuilding; contract incentives.
Real-time performance shortfalls	simulation; benchmarking; prototyping; tuning; technical analysis.
Development technically too difficult	technical analysis; cost–benefit analysis; prototyping; staff training and development.

This top ten list of software risks is based on one presented by Barry Boehm in his *Tutorial on Software Risk Management*, IEEE Computer Society, 1989.

7.7 Evaluating risks to the schedule

We have seen that not all risks can be eliminated – even those that are classified as avoidable or manageable can, in the event, still cause problems affecting activity durations. By identifying and categorizing those risks, and in particular, their likely effects on the duration of planned activities, we can assess what impact they are likely to have on our activity plan.

We will now take a look at two methods for assessing the effects of these uncertainties on the project schedule.

Using PERT to evaluate the effects of uncertainty

PERT was developed to take account of the uncertainty surrounding estimates of task durations. It was developed in an environment of expensive, high-risk and

state-of-the-art projects – not that dissimilar to many of today's large software projects.

PERT (program evaluation and review technique) was published in the same year as CPM. Developed for the Fleet Ballistic Missiles Program it is said to have saved considerable time in development of the Polaris missile.

- **Most likely time** – the time we would expect the task to take under normal circumstances. We shall denote this by the letter m .
- **Optimistic time** – the shortest time in which we could expect to complete the activity, barring outright miracles. We shall use the letter a to denote this.
- **Pessimistic time** – the worst possible time allowing for all reasonable eventualities but excluding 'acts of God and warfare' (as they say in most insurance exclusion clauses). We shall denote this by b .

PERT then combines these three estimates to form a single expected duration, t_e , using the formula

$$t_e = \frac{a + 4m + b}{6}$$

Exercise 7.4

Table 7.3 provides additional activity duration estimates for the network shown in Figure 6.17. There are new estimates for a and b and the original activity duration estimates have been used as the most likely times, m . Calculate the expected duration, t_e , for each activity.

Using expected durations

The expected durations are used to carry out a forward pass through a network; using the same method as the CPM technique. In this case, however, the calculated event dates are not the earliest possible dates but are the dates by which we expect to achieve those events.

Exercise 7.5

Before reading further, use your calculated expected activity durations to carry out a forward pass through the network (Figure 6.17) and verify that the project duration is 13.5 weeks.

What does an expected duration of 13.5 weeks mean in terms of the completion date for the project?

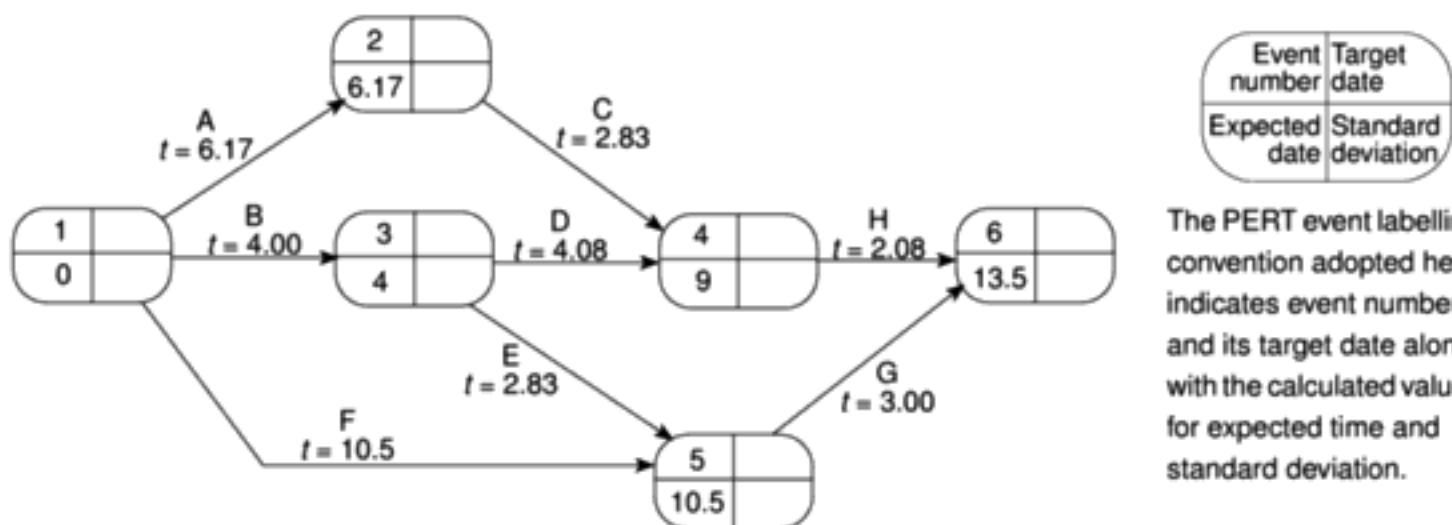
The PERT network illustrated in Figure 7.3 indicates that we expect the project to take 13.5 weeks – unlike CPM, this does not indicate the earliest date by which we could complete the project but the expected (or most likely) date. An advantage of this approach is that it places an emphasis on the uncertainty of the real world.

Table 7.3 PERT activity time estimates

Activity	Activity durations (weeks)		
	Optimistic (a)	Most likely (m)	Pessimistic (b)
A	5	6	8
B	3	4	5
C	2	3	3
D	3.5	4	5
E	1	3	4
F	8	10	15
G	2	3	4
H	2	2	2.5

Rather than being tempted to say ‘the completion date for the project is ...’ we are lead to say ‘we expect to complete the project by ...’.

It also focuses attention on the uncertainty of the estimation of activity durations. Requesting three estimates for each activity emphasizes the fact that we are not certain what will happen – we are forced to take into account the fact that estimates are approximate.

**Figure 7.3** The PERT network after the forward pass.

Activity standard deviations

This standard deviation formula is based on the rationale that there are approximately six standard deviations between the extreme tails of many statistical distributions.

A quantitative measure of the degree of uncertainty of an activity duration estimate may be obtained by calculating the standard deviation s of an activity time, using the formula

$$s = \frac{b - a}{6}$$

The activity standard deviation is proportional to the difference between the optimistic and pessimistic estimates, and can be used as a ranking measure of the degree of uncertainty or risk for each activity. The activity expected durations and standard deviations for our sample project are shown in Table 7.4.

Table 7.4 *Expected times and standard deviations*

Activity	Activity durations (weeks)				
	Optimistic (a)	Most likely (m)	Pessimistic (b)	Expected (t_e)	Standard deviation (s)
A	5	6	8	6.17	0.50
B	3	4	5	4.00	0.33
C	2	3	3	2.83	0.17
D	3.5	4	5	4.08	0.25
E	1	3	4	2.83	0.50
F	8	10	15	10.50	1.17
G	2	3	4	3.00	0.33
H	2	2	2.5	2.08	0.08

The likelihood of meeting targets

The main advantage of the PERT technique is that it provides a method for estimating the probability of meeting or missing target dates. There might be only a single target date – the project completion – but we might wish to set additional intermediate targets.

Suppose that we must complete the project within 15 weeks at the outside. We expect it will take 13.5 weeks but it could take more or, perhaps, less. In addition, suppose that activity C must be completed by week 10, as it is to be carried out by a member of staff who is scheduled to be working on another project and that event 5 represents the delivery of intermediate products to the customer. These three target dates are shown on the PERT network in Figure 7.4.

The PERT technique uses the following three-step method for calculating the probability of meeting or missing a target date:

- calculate the standard deviation of each project event;
- calculate the z value for each event that has a target date;
- convert z values to a probabilities.

Calculating the standard deviation of each project event

Standard deviations for the project events can be calculated by carrying out a forward pass using the activity standard deviations in a manner similar to that used with expected durations. There is, however, one small difference – to add two standard deviations we must add their squares and then find the square root of the sum. Exercise 7.5 illustrates the technique.

The square of the standard deviation is known as the variance. Standard deviations may not be added together but variances may.

The standard deviation for event 3 depends solely on that of activity B. The standard deviation for event 3 is therefore 0.33.

For event 5 there are two possible paths, B + E or F. The total standard deviation for path B + E is $\sqrt{(0.33^2 + 0.50^2)} = 0.6$ and that for path F is 1.17; the standard deviation for event 5 is therefore the greater of the two, 1.17.

Verify that the standard deviations for each of the other events in the project are as shown in Figure 7.4.

Exercise 7.6

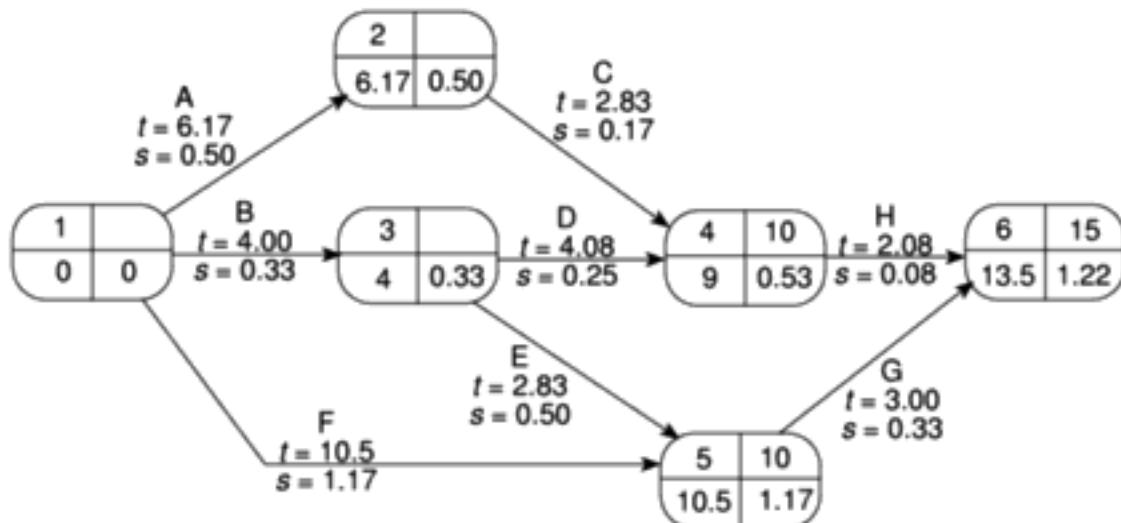


Figure 7.4 The PERT network with three target dates and calculated event standard deviations.

7.8 Calculating the z values

The z value is calculated for each node that has a target date. It is equivalent to the number of standard deviations between the node's expected and target dates. It is calculated using the formula

$$z = \frac{T - t_e}{s}$$

where t_e is the expected date and T the target date.

Exercise 7.7

The z value for event 4 is $(10 - 9.00)/0.53 = 1.8867$.

Calculate the z values for the other events with target dates in the network shown in Figure 7.4.

Converting z values to probabilities

A z value may be converted to the probability of not meeting the target date by using the graph in Figure 7.5.

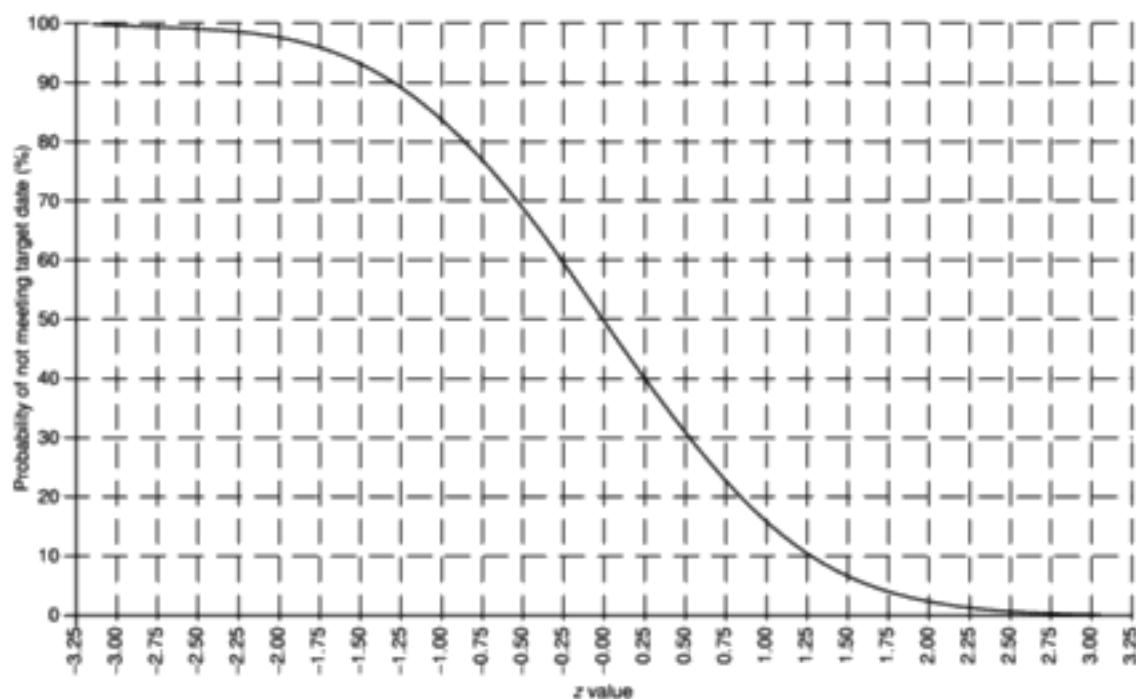


Figure 7.5 The probability of obtaining a value within z standard deviations of the mean for a normal distribution.

This graph is the equivalent of tables of z values, also known as standard normal deviates, which may be found in most statistics textbooks

Exercise 7.8

The z value for the project completion (event 6) is 1.23. Using Figure 7.5 we can see that this equates to a probability of approximately 11%, that is, there is an 11% risk of not meeting the target date of the end of week 15.

Find the probabilities of not achieving events 4 or 5 by their target dates of the end of week 10.

What is the likelihood of completing the project by week 14?

The advantages of PERT

We have seen that by requesting multivalued activity duration estimates and calculating expected dates, PERT focuses attention on the uncertainty of forecasting. We can use the technique to calculate the standard deviation for each

task and use this to rank them according to their degree of risk. Using this ranking, we can see, for example, that activity F is the one over which we have greatest uncertainty, whereas activity C should, in principle, give us relatively little cause for concern.

If we use the expected times and standard deviations for forward passes through the network we can, for any event or activity completion, estimate the probability of meeting any set target. In particular, by setting target dates along the critical path, we can focus on those activities posing the greatest risk to the project's schedule.

Monte Carlo simulation

As an alternative to the PERT technique, and to provide a greater degree of flexibility in specifying likely activity durations, we can use Monte Carlo simulation techniques to evaluate the risks of not achieving deadlines. The basis of this technique is the calculation of event times for a project network a large number of times, each time selecting activity times randomly from a set of estimates for each activity. The results are then be tabulated, summarized or displayed as a graph such as that shown in Figure 7.6.

Monte Carlo simulation was also discussed in Section 3.7 in the context of project evaluation.

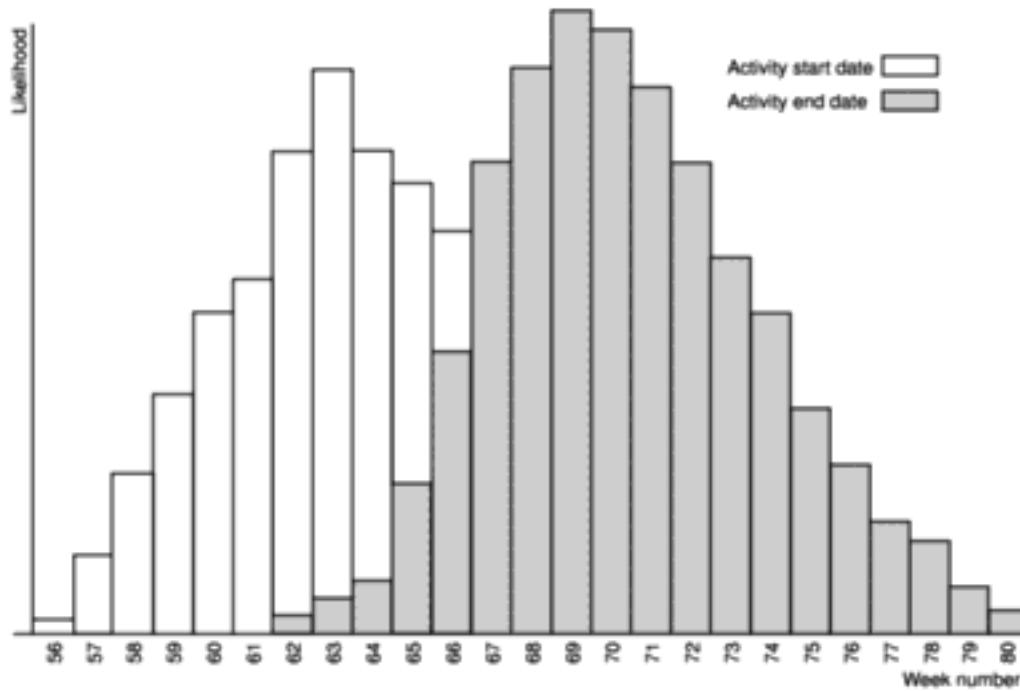


Figure 7.6 Risk profile for an activity generated using Monte Carlo simulation.

Activity duration estimates can be specified in a variety of forms, depending upon the information available. If for example, we have historic data available about the durations of similar activities, we might be able to specify durations as a probability distribution. With less information available we should, at least, be able to provide three time estimates as used by PERT.

There are a number of packages available for carrying out Monte Carlo simulation. Some will exchange data with project scheduling applications and some interface to standard spreadsheet software. The majority of these packages will apply Monte Carlo risk analysis to cost and resource as well as duration estimates.

7.9 Conclusions

In this chapter, we have seen how to identify and manage the risks that might affect the success of a project. Risk management is concerned with assessing and prioritizing risks and drawing up plans for addressing those risks before they become problems.

This chapter has also described techniques for estimating the effect of risk on the project's activity network and schedule.

Many of the risks affecting software projects can be reduced by allocating more experienced staff to those activities that are affected. In the next chapter we consider the allocation of staff to activities in more detail.

7.10 Further exercises

1. Identify five risks than might affect the success of the Brightmouth College payroll project and suggest strategies that Brigette might consider for dealing with each of them.
2. The list of risks and risk reduction strategies in Table 7.2 is concerned with generic risks for software projects. What additional risks, in addition to those itemized in Table 7.1, can you identify that would be specific to Amanda's IOE accounts project?
3. List the major risks that might affect your next programming assignment and identify strategies for minimizing each of those risks.
4. If you have access to a project planning computer application find out whether or not it supports the PERT methods described in this chapter.

Chapter 8

Resource allocation

OBJECTIVES

After completing this chapter you will be able to:

- identify the resources required for a project;
 - make the demand for resources more even throughout the life of a project;
 - produce a work plan and resource schedule.
-

8.1 Introduction

In Chapter 6, we saw how to use activity network analysis techniques to plan *when* activities should take place. This was calculated as a time-span during which an activity should take place – bounded by the earliest start and latest finish dates. In Chapter 7 we used the PERT technique to forecast a range of expected dates by which activities would be completed. In both cases these plans took no account of the availability of resources.

In this chapter we shall see how to match the activity plan to available resources and, where necessary, assess the efficacy of changing the plan to fit the resources. Figure 8.1 shows where resource allocation is applied in Step Wise.

In general, the allocation of resources to activities will lead us to review and modify the ideal activity plan. It may cause us to revise stage or project completion dates. In any event, it is likely to lead to a narrowing of the time-spans within which activities may be scheduled.

The final result of resource allocation will normally be a number of schedules including:

- **activity schedule** indicating the planned start and completion dates for each activity;
- **resource schedule** showing the dates on which each resource will be required and the level of that requirement;
- **cost schedule** showing the planned cumulative expenditure incurred by the use of resources over time.

These schedules will provide the basis for the day-to-day control and management of the project. These are described in Chapter 9.

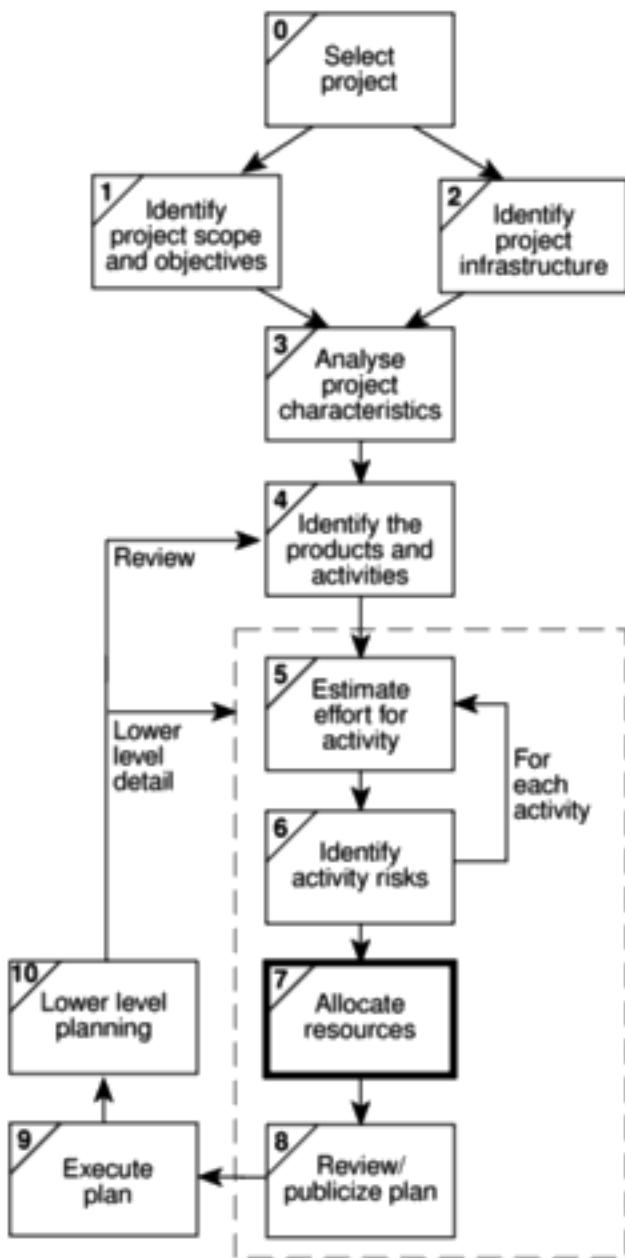


Figure 8.1 Resource allocation is carried out as Step 7.

8.2 The nature of resources

A resource is any item or person required for the execution of the project. This covers many things – from paper clips to key personnel – and it is unlikely that we would wish to itemize every resource required, let alone draw up a schedule for their use! Stationery and other standard office supplies, for example, need not normally be the concern of the project manager – ensuring there is always an adequate supply is the role of the office manager. The project manager must concentrate on those resources where there is a possibility that, without planning, they might not be sufficiently available when required.

Some resources, such as a project manager, will be required for the duration of the project whereas others, such as a specific software developer, might be

required for a single activity. The former, while vital to the success of the project, does not require the same level of scheduling as the latter. Individual programmers, for example, might be committed to working on a number of projects and it will be important to book their time well in advance.

In general, resources will fall into one of seven categories.

- **Labour** The main items in this category will be members of the development project team such as the project manager, systems analysts and software developers. Equally important will be the quality assurance team and other support staff and any employees of the client organization who might be required to undertake or participate in specific activities.
- **Equipment** Obvious items will include workstations and other computing and office equipment. We must not forget that staff also need basic equipment such as desks and chairs.
- **Materials** Materials are items that are consumed, rather than equipment that is used. They are of little consequence in most software projects but can be important for some – software that is to be widely distributed might, for example, require supplies of floppy disks to be specially obtained.
- **Space** For projects that are undertaken with existing staff, space is normally readily available. If any additional staff (recruited or contracted) should be needed then office space will need to be found.
- **Services** Some projects will require procurement of specialist services – development of a wide area distributed system, for example, requires scheduling of telecommunications services.
- **Time** Time is the resource that is being offset against the other primary resources – project time-scales can sometimes be reduced by increasing other resources and will almost certainly be extended if they are unexpectedly reduced.
- **Money** Money is a secondary resource – it is used to buy other resources and will be consumed as other resources are used. It is similar to other resources in that it is available at a cost – in this case interest charges.

The cost of money as a resource is a factor taken into account in DCF evaluation.

8.3 Identifying resource requirements

The first step in producing a resource allocation plan is to list the resources that will be required along with the expected level of demand. This will normally be done by considering each activity in turn and identifying the resources required. It is likely, however, that there will also be resources required that are not activity specific but are part of the project's infrastructure (such as the project manager) or required to support other resources (office space, for example, might be required to house contract software developers).

Case study example

Amanda has produced a precedence network for the IOE project (Figure 8.2) and used this as a basis for a resource requirements list, part of which is shown in Table 8.1. Notice that, at this stage, she has not allocated individuals to tasks but has decided on the type of staff that will be required. The activity durations assume that they will be carried out by 'standard' analysts or software developers.

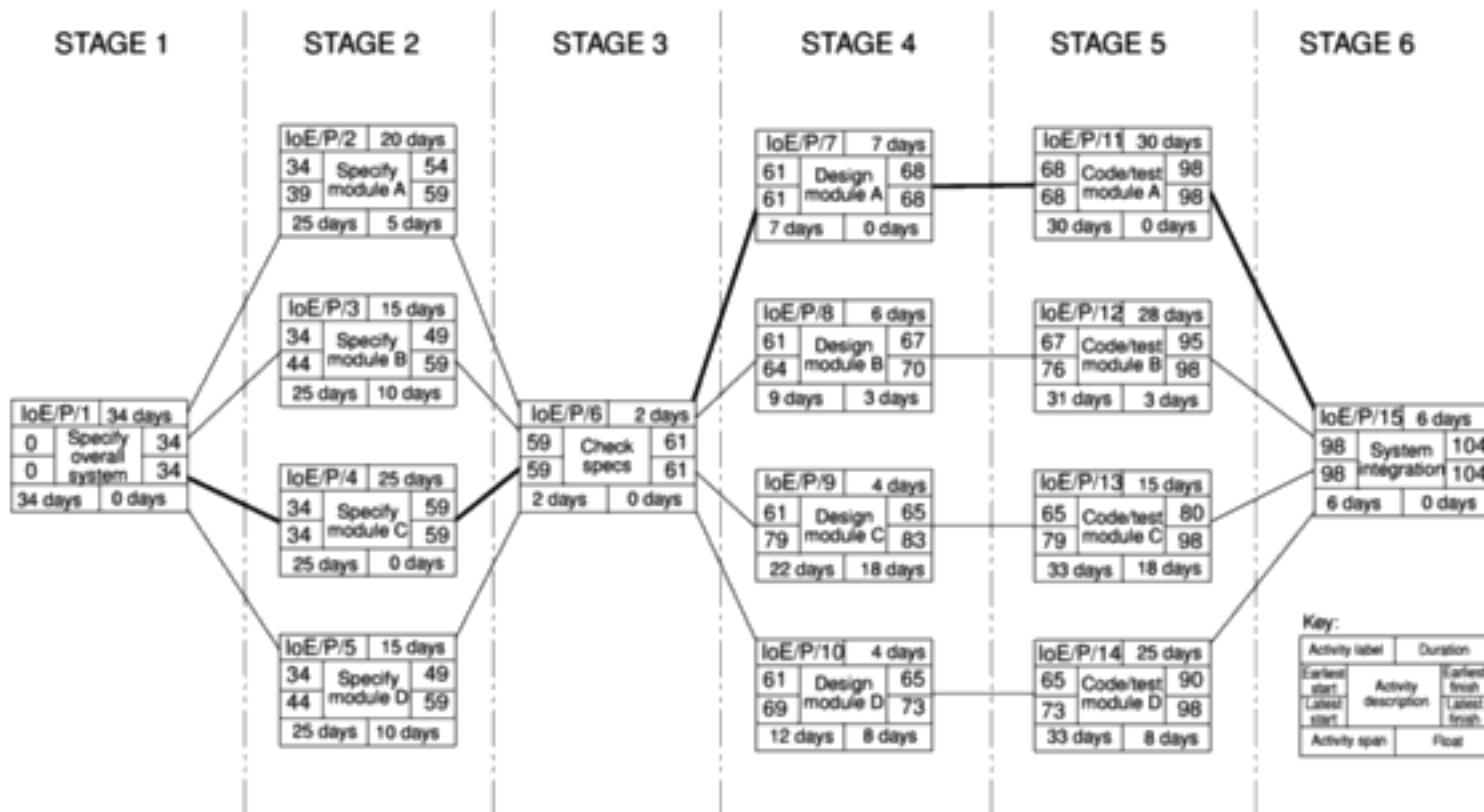


Figure 8.2 The IOE precedence network.

At this stage, it is necessary that the resource requirements list be as comprehensive as possible – it is better that something is included that may later be deleted as unnecessary than to omit something essential. Amanda has therefore included additional office space as a possible requirement, should contract software development staff be recruited.

8.4 Scheduling resources

Having produced the resource requirements list, the next stage is to map this onto the activity plan to assess the distribution of resources required over the duration of the project. This is best done by representing the activity plan as a bar chart and using this to produce a resource histogram for each resource.

Figure 8.3 illustrates Amanda's activity plan as a bar chart and a resource histogram for analyst-designers. Each activity has been scheduled to start at its earliest start date – a sensible initial strategy, since we would, other things being

Table 8.1 Part of Amanda's resource requirements list

<i>Stage</i>	<i>Activity</i>	<i>Resource</i>	<i>Days</i>	<i>Quantity</i>	<i>Notes</i>
ALL		Project manager	104 F/T		
1	All	Workstation	34		Check software availability
	IoE/P/1	Senior analyst	34 F/T		
2	All	Workstation	—	3	1 per person would be ideal
	IoE/P/2	Analyst-designer	20 F/T		
	IoE/P/3	Analyst-designer	15 F/T		
	IoE/P/4	Analyst-designer	25 F/T		
	IoE/P/5	Analyst-designer	15 F/T		Could use analyst-programmer
3	All	Workstation	2 F/T		
	IoE/P/6	Senior analyst*	2 F/T		
4	All	Workstation	—	3	As stage 2
	IoE/P/7	Analyst-designer	7 F/T		
	IoE/P/8	Analyst-designer	6 F/T		
	IoE/P/9	Analyst-designer	4 F/T		
	IoE/P/10	Analyst-designer	4 F/T		
5	All	Workstation	—	4	1 per programmer
	All	Office space	—		If contract programmers used
	IoE/P/11	Programmer	30 F/T		
	IoE/P/12	Programmer	28 F/T		
	IoE/P/13	Programmer	15 F/T		
	IoE/P/14	Programmer	25 F/T		
6	All	Full machine access	—		Approx. 16 hours for full system test
	IoE/P/15	Analyst-designer	6 F/T		

* In reality, this would normally be done by a review involving all the analysts working on stage 2.

equal, wish to save any float to allow for contingencies. Earliest start date scheduling, as is the case with Amanda's project, frequently creates resource histograms that start with a peak and then tail off.

Changing the level of resources on a project over time, particularly personnel, generally adds to the cost of a project. Recruiting staff has costs and even where staff are transferred internally, time will be needed for familiarization with the new project environment.

The resource histogram in Figure 8.3 poses particular problems in that it calls for two analyst-designers to be idle for eleven days, one for six days and one for two days between the specification and design stage. It is unlikely that IOE would have another project requiring their skills for exactly those periods of time and this raises the question as to whether this idle time should be charged to Amanda's

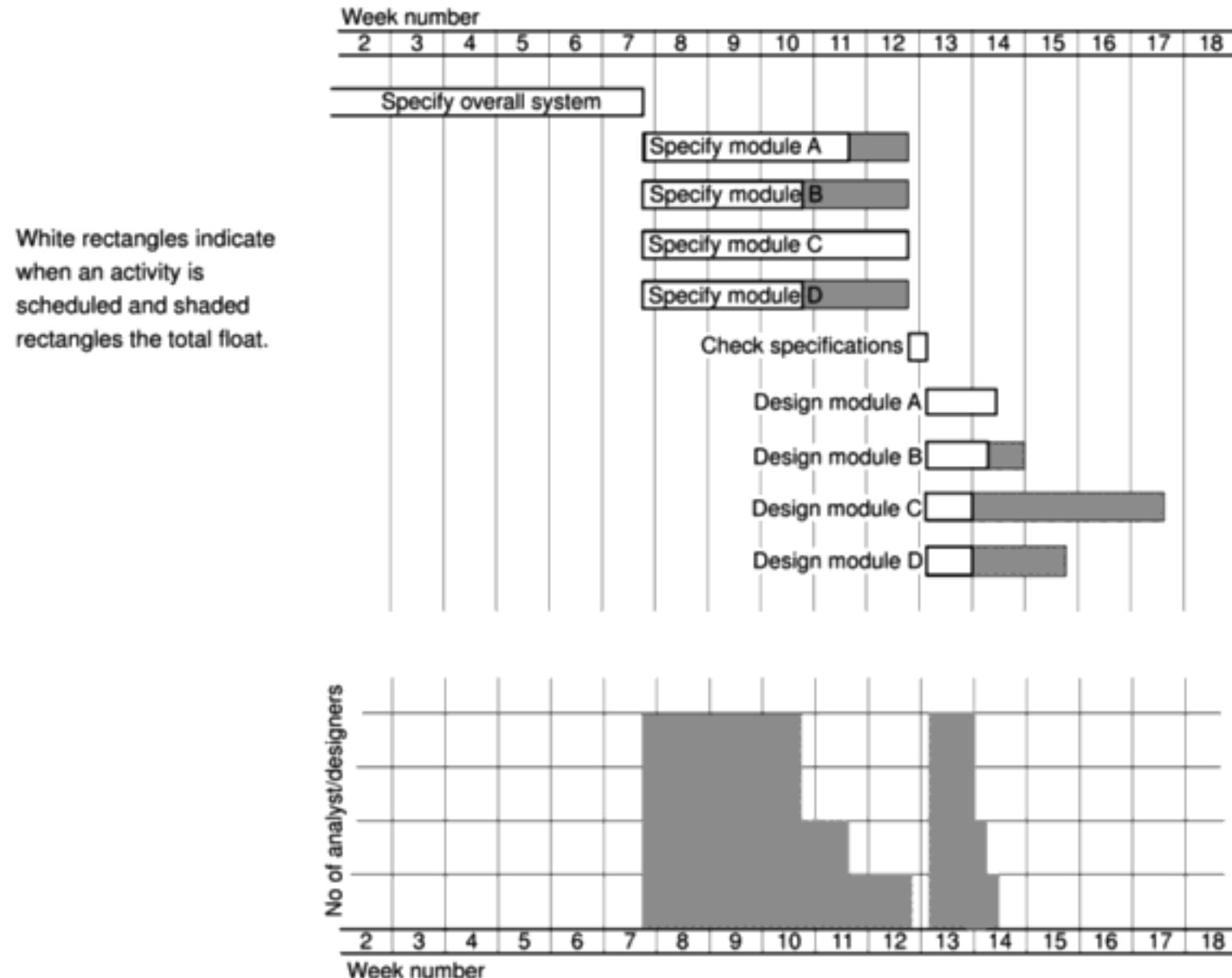
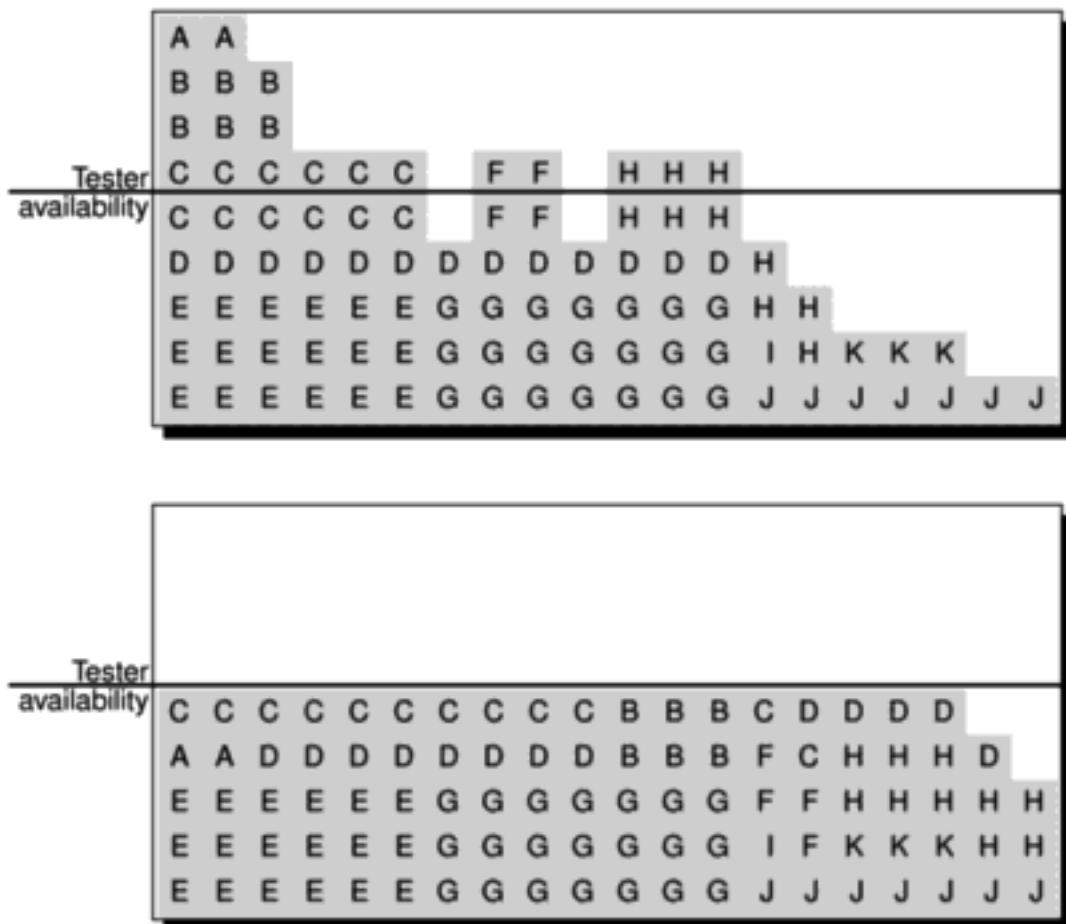


Figure 8.3 Part of Amanda's bar chart and resource histogram for analyst-designers.

project. The ideal resource histogram will be smooth with, perhaps an initial build-up and a staged run-down.

An additional problem with an uneven resource histogram is that it is more likely to call for levels of resource beyond those available. Figure 8.4 illustrates how, by adjusting the start date of some activities and splitting others, a resource histogram can, subject to constraints such as precedence requirements, be smoothed to contain resource demand at available levels. The different letters represent staff working on a series of module testing tasks, that is, one person working on task A, two on tasks B and C etc.

In Figure 8.4, the original histogram was created by scheduling the activities at their earliest start dates. The resource histogram shows the typical peaked shape



The majority of project planning software applications will produce resource histograms based on earliest activity start dates.

Some project planning applications will carry out resource smoothing automatically, although they are unlikely to take into account all the factors that could be used by a project manager

Figure 8.4 A resource histogram showing demand for staff before and after smoothing.

caused by earliest start date scheduling and calls for a total of nine staff where only five are available for the project.

By delaying the start of some of the activities, it has been possible to smooth the histogram and reduce the maximum level of demand for the resource. Notice that some activities, such as C and D, have been split. Where non-critical activities can be split they can provide a useful way of filling troughs in the demand for a resource, but in software projects it is difficult to split tasks without increasing the time they take.

Some of the activities call for more than one unit of the resource at a time – activity F, for example, requires two programmers, each working for two weeks. It might be possible to reschedule this activity to use one programmer over four weeks although that has not been considered in this case.

Amanda has already decided to use only three analyst-designers on the project in order to reduce costs. Her current resource histogram, however, calls for four during both stage 2 and stage 4. Suggest what she might do to smooth the histogram and reduce the number of analyst-designers required to three.

Exercise 8.1

In practice, resources have to be allocated to a project on an activity-by-activity basis and finding the 'best' allocation can be time consuming and difficult. As soon as a member of the project team is allocated to an activity that activity acquires a scheduled start and finish date and the team member becomes unavailable for other activities for that period. Thus, allocating a resource to one activity limits the flexibility for resource allocation and scheduling of other activities.

It is therefore helpful to prioritize activities so that resources can be allocated to competing activities in some rational order. The priority must always be to allocate resources to critical path activities and then to those activities that are most likely to affect others. In that way, lower priority activities are made to fit around the more critical, already scheduled activities.

Of the various ways of prioritizing activities, two are described below.

- **Total float priority** Activities are ordered according to their total float, those with the smallest total float having the highest priority. In the simplest application of this method, activities are allocated resources in ascending order of total float. However, as scheduling proceeds, activities will be delayed (if resources are not available at their earliest start dates) and total floats will be reduced. It is therefore desirable to recalculate floats (and hence reorder the list) each time an activity is delayed.
- **Ordered list priority** With this method, activities that can proceed at the same time are ordered according to a set of simple criteria. An example of this is Burman's priority list, which takes into account activity duration as well as total float:

1. shortest critical activity;
2. critical activities;
3. shortest non-critical activity;
4. non-critical activity with least float;
5. non-critical activities.

Unfortunately, resource smoothing, or even containment of resource demand to available levels, is not always possible within planned time-scales – deferring activities to smooth out resource peaks often puts back project completion. Where that is the case, we need to consider ways of increasing the available resource levels or altering working methods.

P. J. Burman, *Precedence Networks for Planning and Control*, McGraw-Hill, 1972.

Exercise 8.2

Amanda finds that, with only three analyst-designers the specification of module D (see Figure 8.3) will have to be deferred until after the specification of module B and this will add five days to the overall project duration (making 109 in total). She had hoped to have the project completed within 100 days and this is a further disappointment. She therefore decides to have another look at her activity plan.

You will remember that early on she decided that she should check all of the specifications together (activity IoE/P/6) before allowing design to start. It is now apparent that this is causing a significant bottleneck and delaying module D will only exacerbate the problem. She therefore decides on a compromise – she will check the specifications for modules A, B and D together but will then go ahead with their design without waiting for the module C specification. This will be checked against the others when it is complete.

She redraws her precedence network to reflect this, inserting the new activity of checking the module C specification against the others (activity IoE/P/6a). This is shown in Figure 8.5. Draw a new resource histogram to reflect this change.

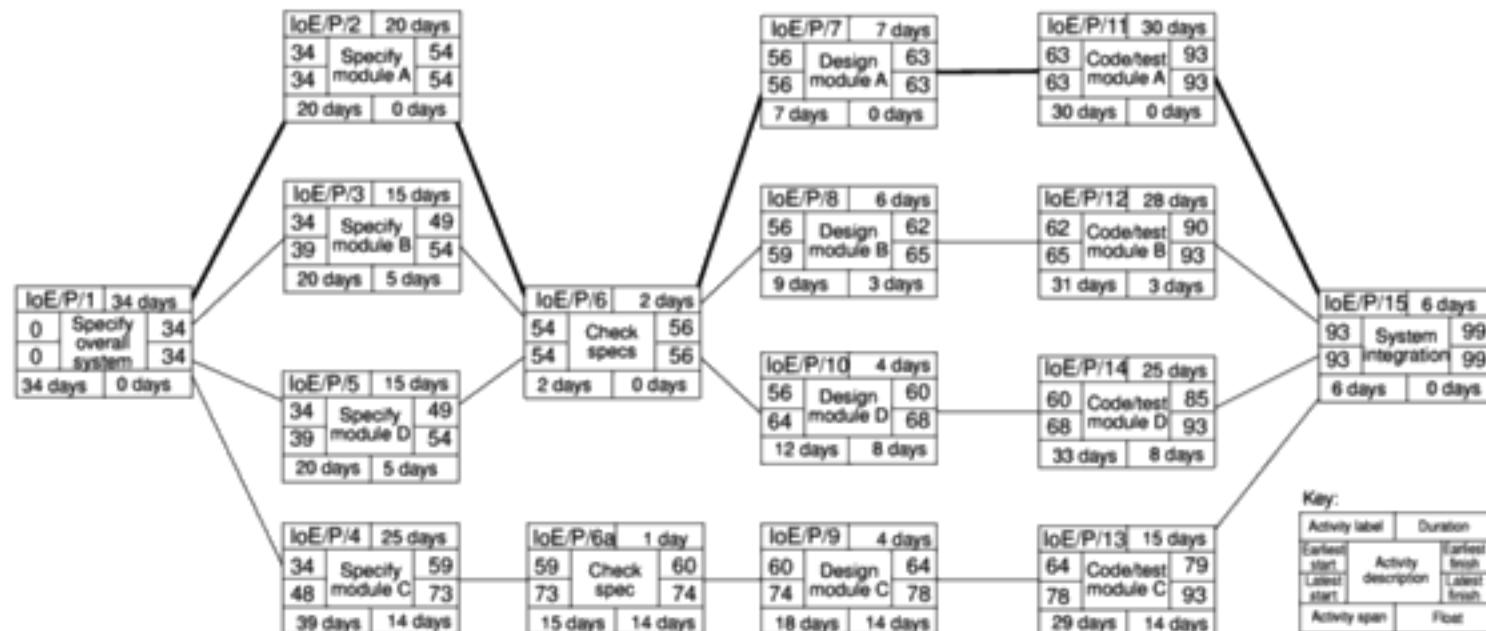


Figure 8.5 Amanda's revised precedence network.

8.5 Creating critical paths

Scheduling resources can create new critical paths. Delaying the start of an activity because of lack of resources will cause that activity to become critical if this uses up its float. Furthermore, a delay in completing one activity can delay the availability of a resource required for a later activity. If the later one is already critical then the earlier one might now have been made critical by linking their resources.

Amanda's revised schedule, which still calls for four analyst-designers but only for a single day, is illustrated in the solution to Exercise 8.2 (check it in the back of the book if you have not done so already). Notice that in rescheduling some of the activities she has introduced additional critical activities. Delaying the specification of module C has used up all of its float – and that of the subsequent

activities along that path! Amanda now has two critical paths – the one shown on the precedence network and the new one.

In a large project, resource-linked criticalities can be quite complex – a hint of the potential problems may be appreciated by looking at the next exercise.

Exercise 8.3

Amanda decides to delay the specification of module C for a further day to ensure that only three analyst-designers will be required. The relevant part of her revised bar chart and resource histogram are shown in Figure 8.6.

Which activities will now be critical?

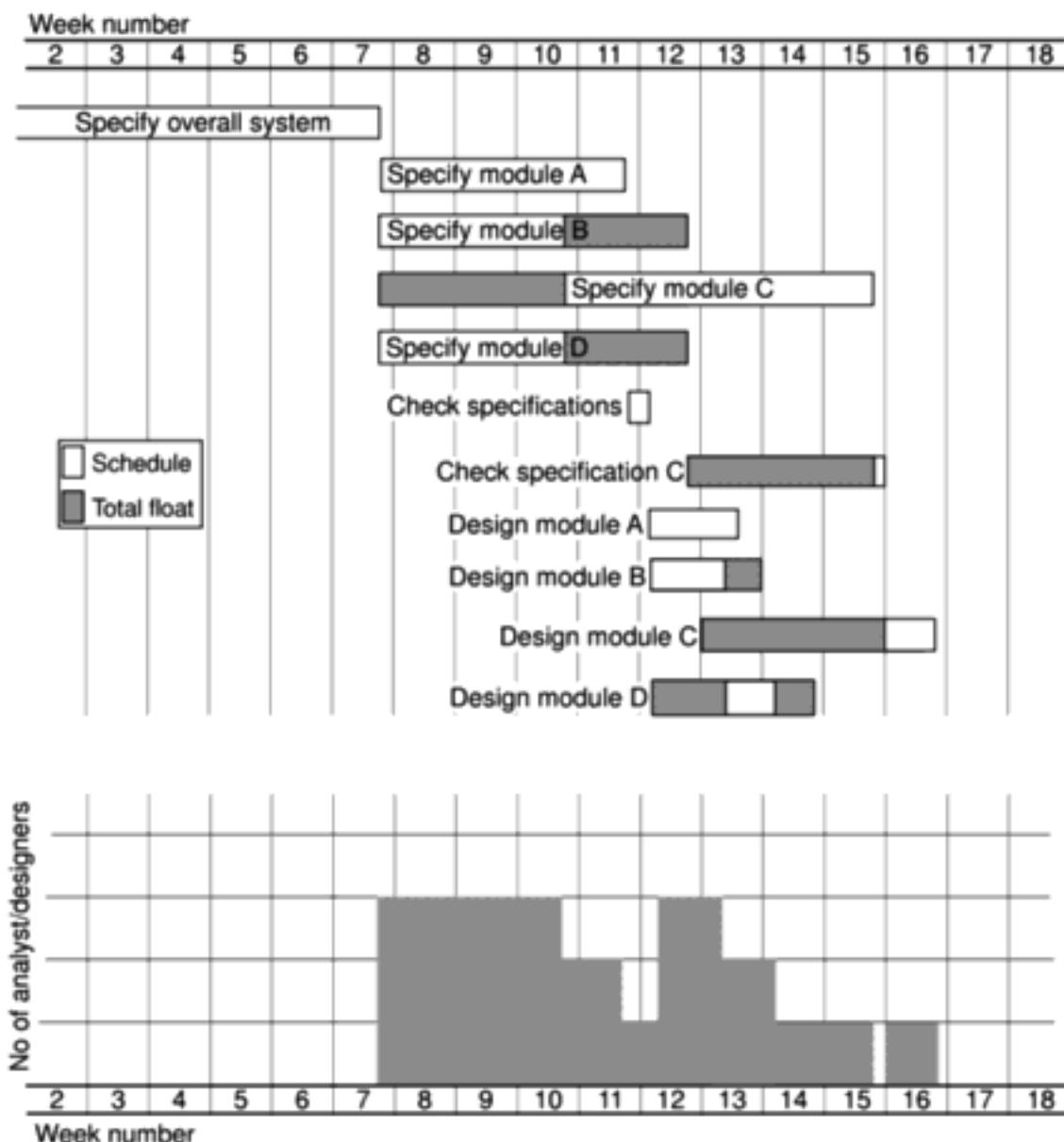


Figure 8.6 Amanda's project scheduled to require three analyst-designers.

8.6 Counting the cost

The discussion so far has concentrated on trying to complete the project by the earliest completion date with the minimum number of staff. We have seen that doing this places constraints on when activities can be carried out and increases the risk of not meeting target dates.

Alternatively, Amanda could have considered using additional staff or lengthening the overall duration of the project. The additional costs of employing extra staff would need to be compared to the costs of delayed delivery and the increased risk of not meeting the scheduled date. The relationship between these factors is discussed later in this chapter.

8.7 Being specific

Allocating resources and smoothing resource histograms is relatively straightforward where all resources of a given type can be considered more or less equivalent. When allocating labourers to activities in a large building project we need not distinguish among individuals – there are likely to be many labourers and they may be treated as equals so far as skills and productivity are concerned.

This is seldom the case with software projects. We saw in Chapter 5 that, because of the nature of software development, skill and experience play a significant part in determining the time taken and, potentially, the quality of the final product. With the exception of extremely large projects it makes sense to allocate individual members of staff to activities as early as possible, as this can lead us to revise our estimate of their duration.

In allocating individuals to tasks, a number of factors need to be taken into account.

- **Availability** We need to know whether a particular individual will be available when required. Reference to the departmental work plan determines this but the wise project manager will always investigate the risks that might be involved – earlier projects might, for example, over-run and affect the availability of an individual.
- **Criticality** Allocation of more experienced personnel to activities on the critical path often helps in shortening project durations or at least reduces the risk of overrun.
- **Risk** We saw how to undertake activity risk assessment in the previous chapter. Identifying those activities posing the greatest risk, and knowing the factors influencing them, helps to allocate staff. Allocating the most experienced staff to the highest risk activities is likely to have the greatest effect in reducing overall project uncertainties. More experienced staff are, however, usually more expensive.
- **Training** It will benefit the organization if positive steps are taken to allocate junior staff to appropriate non-critical activities where there will be sufficient

Reappraisal of the critical path and PERT or Monte Carlo risk analysis might need to be carried out in parallel with staff allocation.

slack for them to train and develop skills. There can even be direct benefits to the particular project since some costs may be allocated to the training budget.

- **Team Building** The selection of individuals must also take account of the final shape of the project team and the way they will work together. This and additional aspects of personnel management are discussed in Chapter 11.

Exercise 8.4

Amanda has decided that, where possible, whoever writes the specification for a module should also produce the design, as she believes this will improve the commitment and motivation of the three analyst-designers, Belinda, Tom and Daisy.

She has decided that she will use Tom, a trainee analyst-designer, for the specification and design of module D as both of these activities have a large float compared to their activity span ($\frac{1}{21}$ and $\frac{1}{13}$ of their span respectively). Since the specification and design of module C are on the critical path, she decides to allocate both of these tasks to Belinda, a particularly experienced and capable member of staff.

Having made these decisions she has almost no flexibility in how she assigns the other specification and design activities. Work out from the activity bar chart produced as part of the solution to Exercise 8.2 (shown in Figure 8.6) whom she assigns to which of the remaining specification and design activities.

8.8 Publishing the resource schedule

In allocating and scheduling resources we have used the activity plan (a precedence network in the case of the examples in this chapter), activity bar charts and resource histograms. Although good as planning tools, they are not the best way of publishing and communicating project schedules. For this we need some form of work plan. Work plans are commonly published either as lists or charts such as that illustrated in Figure 8.7. In this case Amanda has chosen not to include activity floats (which could be indicated by shaded bars) as she fears that one or two members of the team might work with less urgency if they are aware that their activities are not critical.

Notice that, somewhat unusually, it is assumed there are no public holidays or other non-productive periods during the 100 days of the project and that none of the team has holidays for the periods they are shown as working.

Amanda has also made no explicit allowance for staff taking sick leave.

Amanda now transfers some of the information from the work schedule to her precedence network. In particular, she amends the earliest start dates for activities and any other constraints (such as revised latest finish dates where resources need to be made available) that have been introduced. A copy of her revised precedence network is shown in Figure 8.8 – notice that she has highlighted all critical activities and paths.

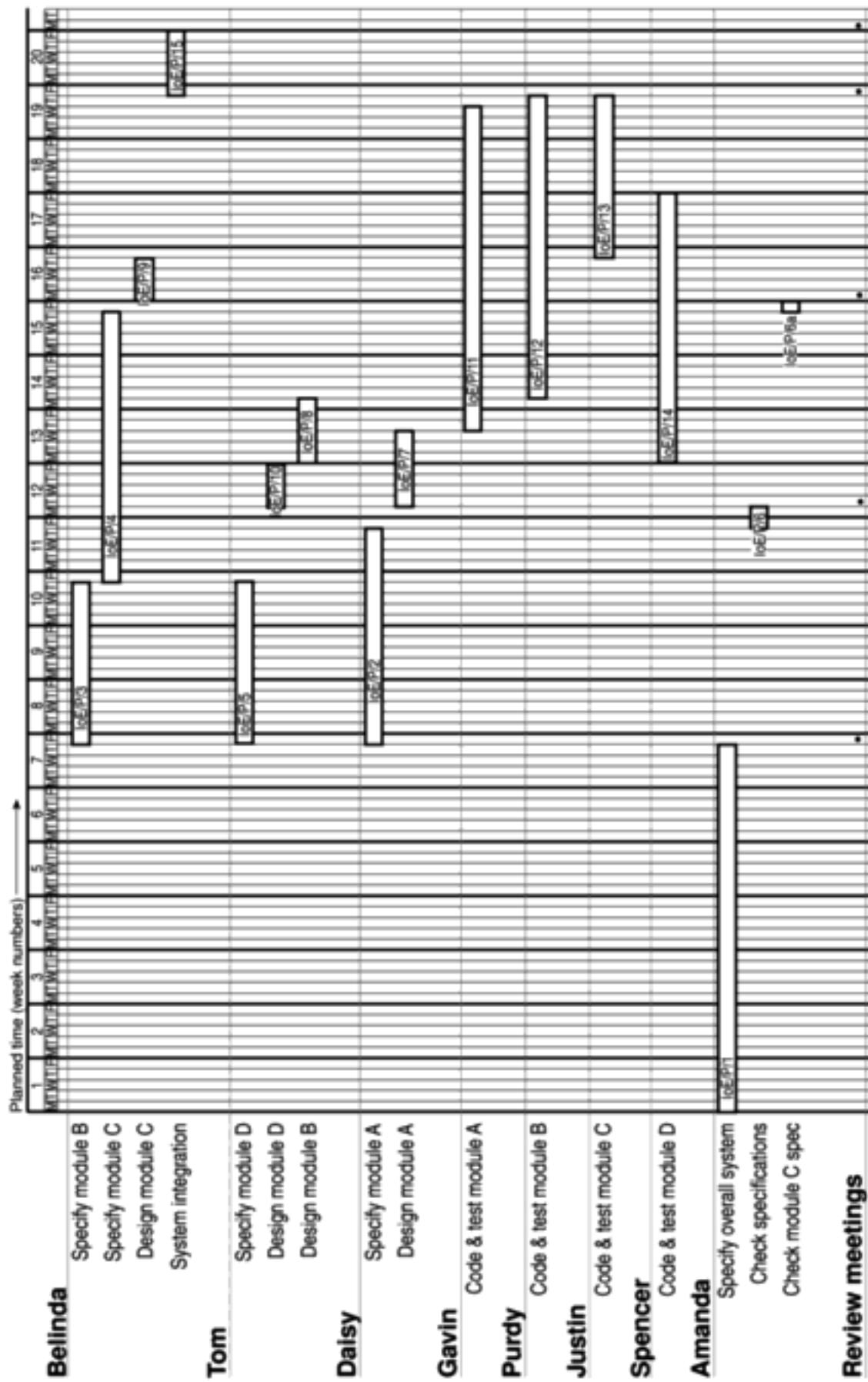


Figure 8.7 Amanda's work schedule.

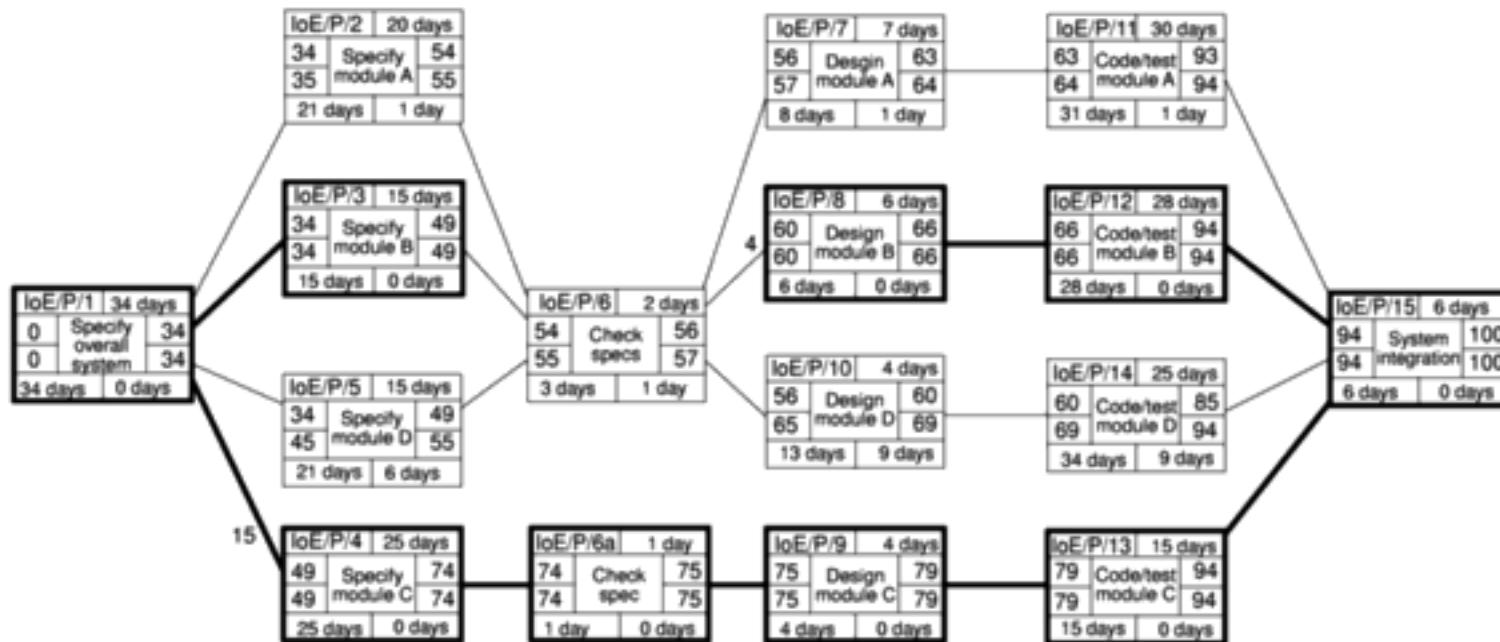


Figure 8.8 Amanda's revised precedence network showing scheduled start and completion dates.

8.9 Cost schedules

It is now time to produce a detailed cost schedule showing weekly or monthly costs over the life of the project. This will provide a more detailed and accurate estimate of costs and will serve as a plan against which project progress can be monitored.

Calculating cost is straightforward where the organization has standard cost figures for staff and other resources. Where this is not the case, then the project manager will have to calculate the costs.

In general, costs are categorized as follows.

- **Staff costs** These will include staff salaries as well as the other direct costs of employment such as the employer's contribution to social security funds, pension scheme contributions, holiday pay and sickness benefit. These are commonly charged to projects at hourly rates based on weekly work records completed by staff. Note that contract staff are usually charged by the week or month – even when they are idle.
- **Overheads** Overheads represent expenditure that an organization incurs, which cannot be directly related to individual projects or jobs including space rental, interest charges and the costs of service departments (such as personnel). Overhead costs can be recovered by making a fixed charge on development departments (in which case they usually appear as a weekly or monthly charge for a project), or by an additional percentage charge on direct staff employment costs. These additional charges or oncosts can easily equal or exceed the direct employment costs.

- Usage charges** In some organizations, projects are charged directly for use of resources such as computer time (rather than their cost being recovered as an overhead). This will normally be on an 'as used' basis.

Amanda finds that IOE recovers some overheads as oncosts on direct staff costs although others are recovered by charging a fixed £200 per day against projects. Staff costs (including overheads) are as shown in Table 8.2. In addition to the commitments in the work plan (Figure 8.7) Amanda estimates that, in total, she will have spent an additional 10 days planning the project and carrying out the post-project review.

Calculate the total cost for Amanda's project on this basis. How is the expenditure spread over the life of the project?

Exercise 8.5

Table 8.2 *Staff costs (including oncosts) for Amanda's project team*

<i>Staff member</i>	<i>Daily cost (£)</i>
Amanda	300
Belinda	250
Tom	175
Daisy	225
Gavin	150
Purdy	150
Justin	150
Spencer	150

Figure 8.9 shows the weekly costs over the 20 weeks that Amanda expects the project to take. This is a typical cost profile – building up slowly to a peak and then tailing off quite rapidly at the end of the project. Figure 8.10 illustrates the cumulative cost of the project and it is generally this that would be used for cost control purposes.

8.10 The scheduling sequence

Going from an ideal activity plan to a costed schedule can be represented as a sequence of steps, rather like the classic waterfall life-cycle model. In the ideal world, we would start with the activity plan and use this as the basis for our risk assessment. The activity plan and risk assessment would provide the basis for our resource allocation and schedule from which we would produce cost schedules.

In practice, as we have seen by looking at Amanda's project, successful resource allocation often necessitates revisions to the activity plan, which, in turn, will affect our risk assessment. Similarly, the cost schedule might indicate the need or

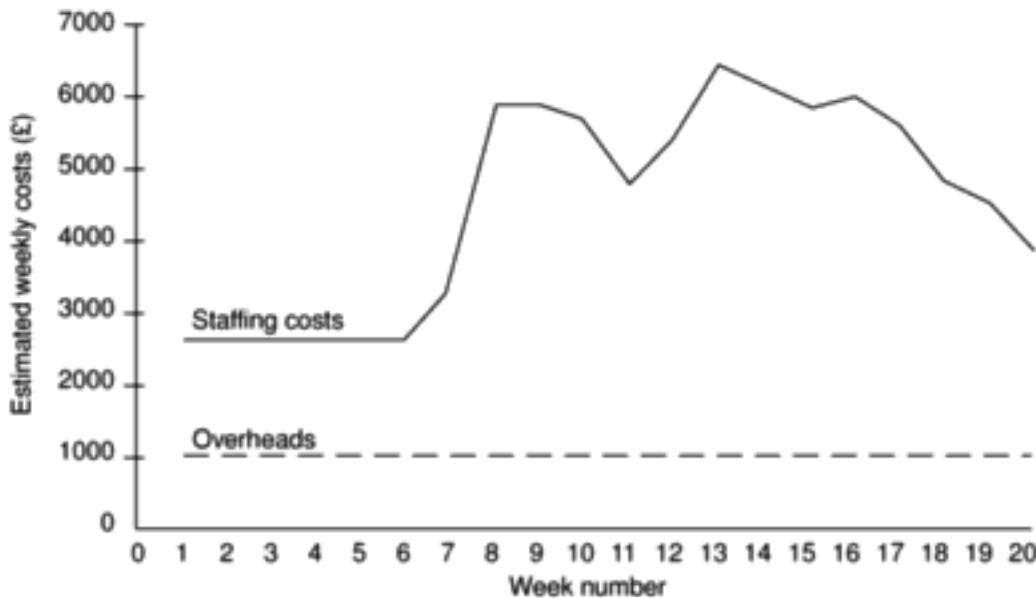


Figure 8.9 Weekly project costs for the IOE project.

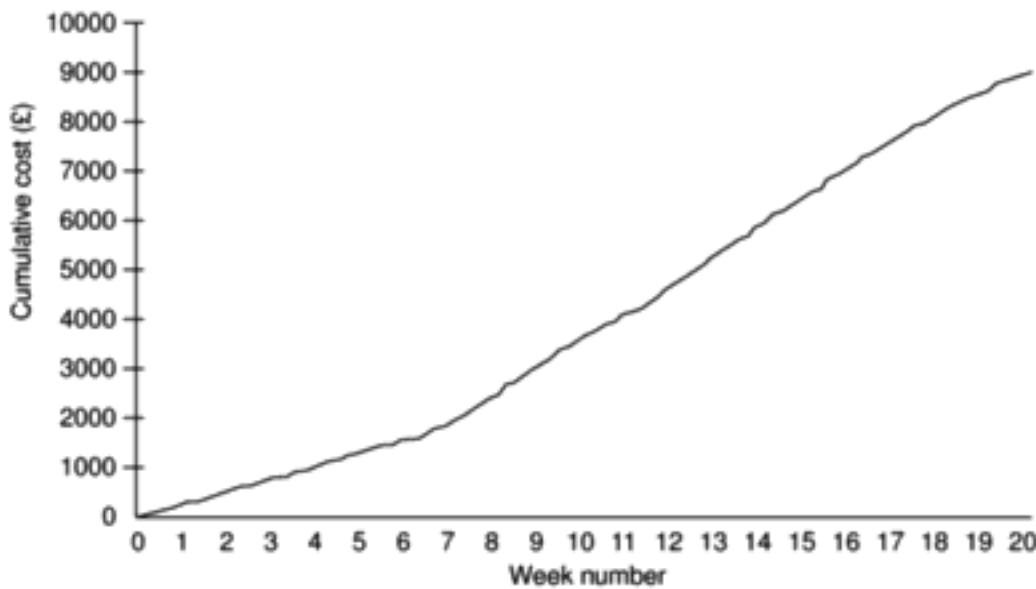


Figure 8.10 Cumulative project costs for the IOE project.

desirability to reallocate resources or revise activity plans – particularly where that schedule indicates a higher overall project cost than originally anticipated.

The interplay between the plans and schedules is complex – any change to any one will affect each of the others. Some factors can be directly compared in terms of money – the cost of hiring additional staff can be balanced against the costs of delaying the project's end date. Some factors, however, are difficult to express in money terms (the cost of an increased risk, for example) and will include an element of subjectivity.

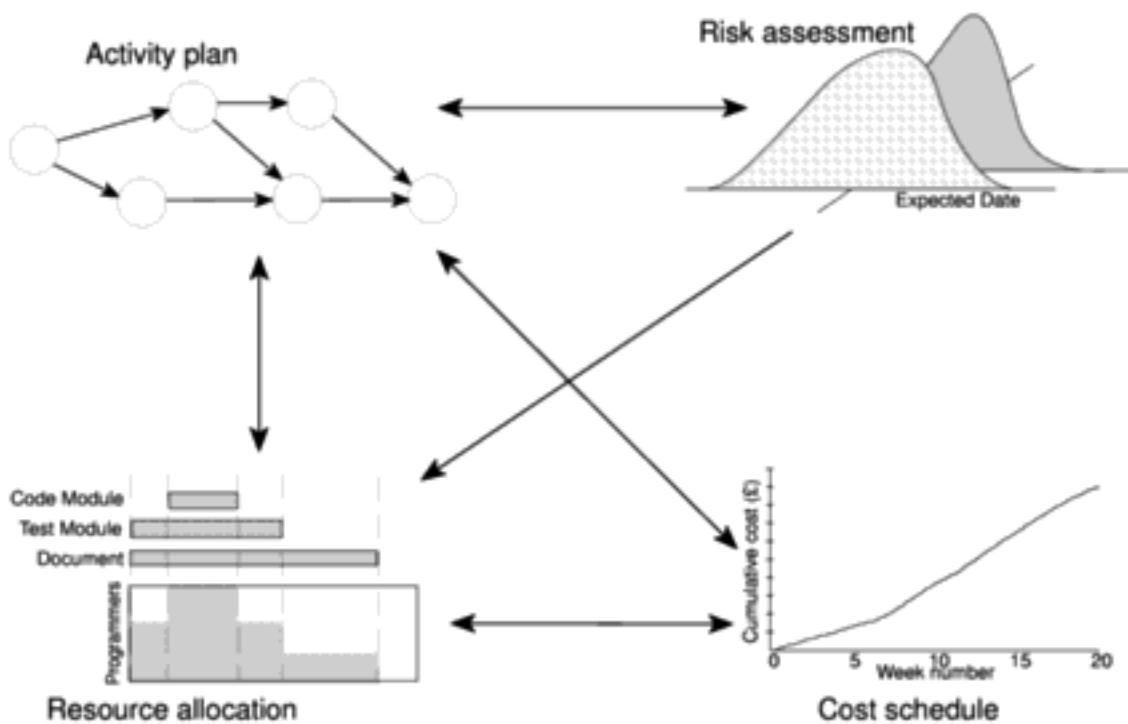


Figure 8.11 Successful project scheduling is not a simple sequence.

While good project planning software will assist greatly in demonstrating the consequences of change and keeping the planning synchronized, successful project scheduling is largely dependant upon the skill and experience of the project manager in juggling the many factors involved.

8.11 Conclusion

In this chapter we have discussed the problems of allocating resources to project activities and the conversion of an activity plan to a work schedule. In particular, we have seen the importance of the following:

- identifying all the resources needed;
- arranging activity starts to minimize variations in resource levels over the duration of the project;
- allocating resources to competing activities in a rational order of priority;
- taking care in allocating the right staff to critical activities.

8.12 Further exercises

1. Burman's priority ordering for allocating resources to activities takes into account the activity duration as well as its total float. Why do you think this is advantageous?

2. If you have access to project planning software use it to produce an activity plan for Amanda's project and include the staff resource requirements for each activity. Explore the facilities of your software and answer the following questions.
 - Can you set up resource types and ask the application to allocate individuals to tasks?
 - Will your software allow you to specify productivity factors for individual members of staff so that the duration of an activity depends upon who is carrying it out?
 - Will your software carry out resource smoothing or provide a minimum cost solution?
 - Can you replicate Amanda's resource schedule (see Figure 8.7) – or produce a better one?
3. On a large project it is often be the responsibility of a team leader to allocate tasks to individuals. Why might it be unsatisfactory to leave such allocations entirely to the discretion of the team leader?
4. In scheduling her project, Amanda ignored the risks of absence due to staff sickness. What might she have done to estimate the likelihood of this occurring and how might she have taken account of the risk when scheduling the project?

Chapter 9

Monitoring and control

OBJECTIVES

When you have completed this chapter you will be able to:

- monitor the progress of projects;
 - assess the risk of slippage;
 - visualize and assess the state of a project;
 - revise targets to correct or counteract drift;
 - control changes to a project's requirements.
-

9.1 Introduction

Once work schedules have been published and the project is under way, attention must be focused on ensuring progress. This requires monitoring of what is happening, comparison of actual achievement against the schedule and, where necessary, revision of plans and schedules to bring the project as far as possible back on target.

In earlier chapters we have stressed the importance of producing plans that can be monitored – for example, ensuring that activities have clearly defined and visible completion points. We will discuss how information about project progress is gathered and what actions must be taken to ensure a project meets its targets.

The final part of this chapter discusses how we can deal with changes that are imposed from outside – namely, changes in requirements.

9.2 Creating the framework

Exercising control over a project and ensuring that targets are met is a matter of regular monitoring, finding out what is happening, and comparing it with current

targets. If there is a mismatch between the planned outcomes and the actual ones then either replanning is needed to bring the project back on target or the target will have to be revised. Figure 9.1 illustrates a model of the project control cycle and shows how, once the initial project plan has been published, project control is a continual process of monitoring progress against that plan and, where necessary, revising the plan to take account of deviations. It also illustrates the important steps that must be taken after completion of the project so that the experience gained in any one project can feed into the planning stages of future projects, thus allowing us to learn from past mistakes.

See Chapter 11 for a discussion of software quality.

In practice we are normally concerned with departures from the plan in four dimensions – delays in meeting target dates, shortfalls in quality, inadequate functionality, and costs going over target. In this chapter we are mainly concerned with the first and last of these.

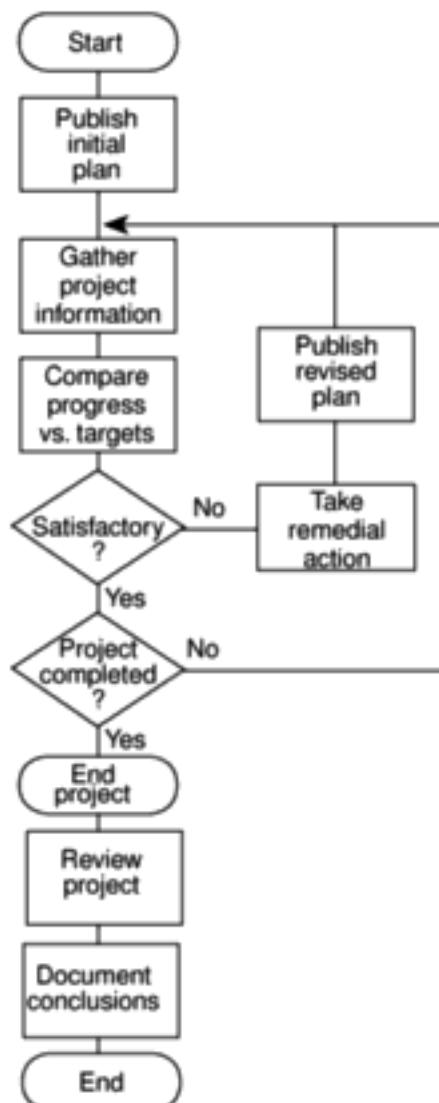


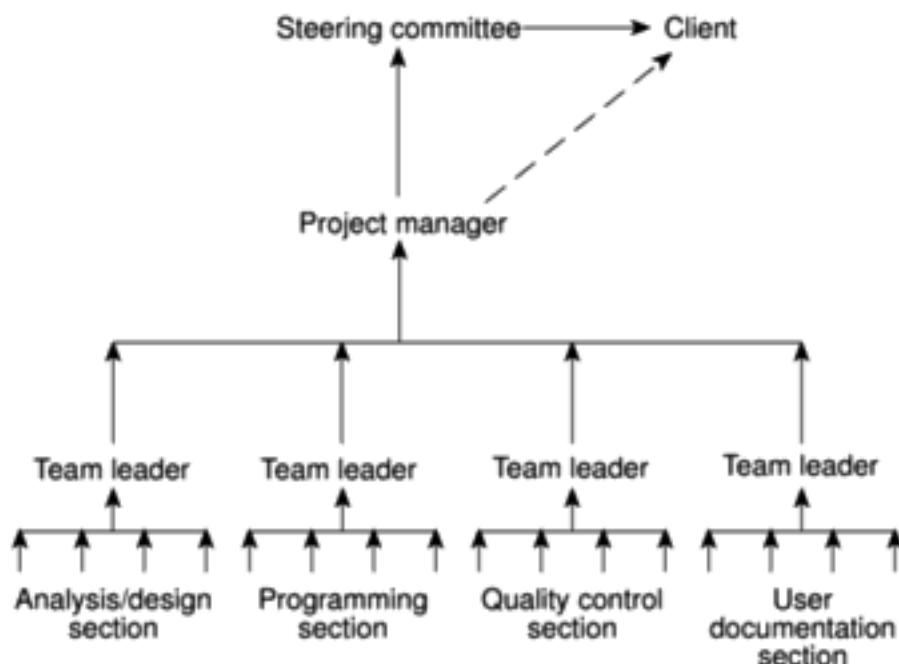
Figure 9.1 The project control cycle.

Responsibility

The overall responsibility for ensuring satisfactory progress on a project is often the role of the *project steering committee* or *Project Board*. Day-to-day responsibility will rest with the project manager and, in all but the smallest of projects, aspects of this can be delegated to team leaders.

Figure 9.2 illustrates the typical reporting structure found with medium and large projects. With small projects (employing around half a dozen or fewer staff) individual team members usually report directly to the project manager, but in most cases team leaders will collate reports on their section's progress and forward summaries to the project manager. These, in turn, will be incorporated into project-level reports for the steering committee and, via them or directly, progress reports for the client.

The concept of a reporting hierarchy was introduced in Chapter 1.



In a PRINCE 2 environment, there is a Project Assurance function reporting to the Project Board and independent of the Project Manager.

Figure 9.2 Project reporting structures.

Reporting may be oral or written, formal or informal, or regular or ad hoc and some examples of each type are given in Table 9.1. While any effective team leader or project manager will be in touch with team members and available to discuss problems, any such informal reporting of project progress must be complemented by formal reporting procedures – and it is those we are concerned with in this chapter.

Assessing progress

Progress assessment will normally be made on the basis of information collected and collated at regular intervals or when specific events occur. Wherever possible, this information will be objective and tangible – whether or not a particular report has been delivered, for example. However, such end-of-activity deliverables might

Table 9.1 Categories of reporting

<i>Report type</i>	<i>Examples</i>	<i>Comment</i>
Oral formal regular	weekly or monthly progress meetings	while reports may be oral formal written minutes should be kept
Oral formal ad hoc	end-of-stage review meetings	while largely oral, likely to receive and generate written reports
Written formal regular	job sheets, progress reports	normally weekly using forms
Written formal ad hoc	exception reports, change reports	
Oral informal ad hoc	canteen discussion, social interaction	often provides early warning; must be backed up by formal reporting

not occur sufficiently frequently throughout the life of the project. Here progress assessment will have to rely on the judgement of the team members who are carrying out the project activities.

Setting checkpoints

It is essential to set a series of checkpoints in the initial activity plan. Checkpoints may be:

- regular (monthly, for example);
- tied to specific events such as the production of a report or other deliverable.

Taking snap-shots

The frequency with which a manager needs to receive information about progress will depend upon the size and degree of risk of the project or that part of the project under their control. Team leaders, for example, need to assess progress daily (particularly when employing inexperienced staff) whereas project managers may find weekly or monthly reporting appropriate. In general, the higher the level, the less frequent and less detailed the reporting needs to be.

There are, however, strong arguments in favour of formal weekly collection of information from staff carrying out activities. Collecting data at the end of each week ensures that information is provided while memories are still relatively fresh and provides a mechanism for individuals to review and reflect upon their progress during the past few days.

Short, Monday morning team progress meetings are a common way of motivating staff to meet short term targets.

Major, or project-level, progress reviews will generally take place at particular points during the life of a project – commonly known as *review points* or *control points*. PRINCE 2, for example, designates a series of checkpoints where the status of work in a project or for a team is reviewed. At the end of each project Stage, PRINCE 2 provides for an End Stage Assessment where an assessment of the project and consideration of its future are undertaken.

9.3 Collecting the data

As a rule, managers will try to break down long activities into more controllable tasks of one or two weeks duration. However, it will still be necessary to gather information about partially completed activities and, in particular, forecasts of how much work is left to be completed. It can be difficult to make such forecasts accurately.

A software developer working on Amanda's project has written the first 250 lines of a Cobol program that is estimated to require 500 lines of code. Explain why it would be unreasonable to assume that the programming task is 50% complete.

Exercise 9.1

How might you make a reasonable estimate of how near completion it might be?

Where there is a series of products, partial completion of activities is easier to estimate. Counting the number of record specifications or screen layouts produced, for example, can provide a reasonable measure of progress.

In some cases, intermediate products can be used as in-activity milestones. The first successful compilation of a Cobol program, for example, might be considered a milestone even though it is not the final product of the activity code and test.

Partial completion reporting

Many organizations use standard accounting systems with weekly time sheets to charge staff time to individual jobs. The staff time booked to a project indicates the work carried out and the charges to the project. It does not, however, tell the project manager what has been produced or whether tasks are on schedule.

It is therefore common to adapt or enhance existing accounting data collection systems to meet the needs of project control. Weekly time sheets, for example, are frequently adapted by breaking jobs down to activity level and requiring information about work done in addition to time spent. Figure 9.3 shows a typical example of such a report form, in this case requesting information about likely slippage of completion dates as well as estimates of completeness.

Asking for estimated completion times can be criticized on the grounds that frequent invitations to reconsider completion dates deflects attention away from the importance of the originally scheduled targets and can generate an ethos that it is acceptable for completion dates to slip.

Weekly timesheets are a valuable source of information about resources used.

They are often used to provide information about what has been achieved. However, requesting partial completion estimates where they cannot be obtained from objective measures encourages the 99% complete syndrome – tasks are reported as on time until 99% complete, and then stay at 99% complete until finished.

Time Sheet						
Staff	John Smith			Week ending	26/3/99	
Rechargeable hours						
Project	Activity code	Description	Hours this week	% Complete	Scheduled completion	Estimated completion
P21	A243	Code mod A3	12	30	24/4/99	24/4/99
P34	B771	Document take-on	20	90	1/4/99	29/3/99
				Total rechargeable hours	32	
Non-rechargeable hours						
Code	Description		Hours	Comment & authorization		
299	day in lieu		8	Authorized by RS		
				Total non-rechargeable hours	8	

Figure 9.3 A weekly time sheet and progress review form.

Risk reporting

One popular way of overcoming the objections to partial completion reporting is to avoid asking for estimated completion dates, but to ask instead for the team members' estimates of the likelihood of meeting the planned target date.

One way of doing this is the traffic-light method. This consists of the following steps:

- identify the key (first level) elements for assessment in a piece of work;
- break these key elements into constituent elements (second level);
- assess each of the second level elements on the scale *green* for 'on target', *amber* for 'not on target but recoverable', and *red* for 'not on target and recoverable only with difficulty';
- review all the second level assessments to arrive at first level assessments;
- review first and second level assessments to produce an overall assessment.

There are a number of variations on the traffic-light technique. The version described here is in use in IBM and is described in Down, Coleman and Absolon, *Risk Management for Software Projects*, McGraw-Hill, 1994.

For example, Amanda decides to use a version of the traffic-light method for reviewing activities on the IOE project. She breaks each activity into a number of component parts (deciding, in this case, that a further breakdown is unnecessary) and gets the team members to complete a return at the end of each week. Figure 9.4 illustrates Justin's completed assessment at the end of week 16.

Activity Assessment Sheet						
Staff	Justin					
Ref: IoE/P/13	Activity: Code & test module C					
Week number	13	14	15	16	17	18
Activity Summary	G	A	A	R		
Component						Comments
Screen handling procedures	G	A	A	G		
File update procedures	G	G	R	A		
Housekeeping procedures	G	G	G	A		
Compilation	G	G	G	R		
Test data runs	G	G	G	A		
Program documentation	G	G	A	R		

Note that this form refers only to uncompleted activities. Justin would still need to report activity completions and the time spent on activities.

Figure 9.4 A traffic-light assessment of IoE/P/13.

Traffic-light assessment highlights only risk of non-achievement; it is not an attempt to estimate work done or to quantify expected delays.

Following completion of assessment forms for all activities, the project manager uses these as a basis for evaluating the overall status of the project. Any critical activity classified as amber or red will require further consideration and often leads to a revision of the project schedule. Non-critical activities are likely to be considered as a problem if they are classified as red, especially if all their float is likely to be consumed.

9.4 Visualizing progress

Having collected data about project progress, a manager needs some way of presenting that data to greatest effect. In this section, we look at some methods of presenting a picture of the project and its future. Some of these methods (such as Gantt charts) provide a static picture, a single snap-shot, whereas others (such as time-line charts) try to show how the project has progressed and changed through time.

The Gantt chart

One of the simplest and oldest techniques for tracking project progress is the Gantt chart. This is essentially an activity bar chart indicating scheduled activity dates

and durations frequently augmented with activity floats. Reported progress is recorded on the chart (normally by shading activity bars) and a 'today cursor' provides an immediate visual indication of which activities are ahead or behind schedule. Figure 9.5 shows part of Amanda's Gantt chart as at the end of Tuesday of week 17. *Code & test module D* has been completed ahead of schedule and *code & test module A* appears also to be ahead of schedule. The coding and testing of the other two modules are behind schedule.

Henry Gantt (1861–1919)
was an industrial engineer interested in the efficient organization of work.

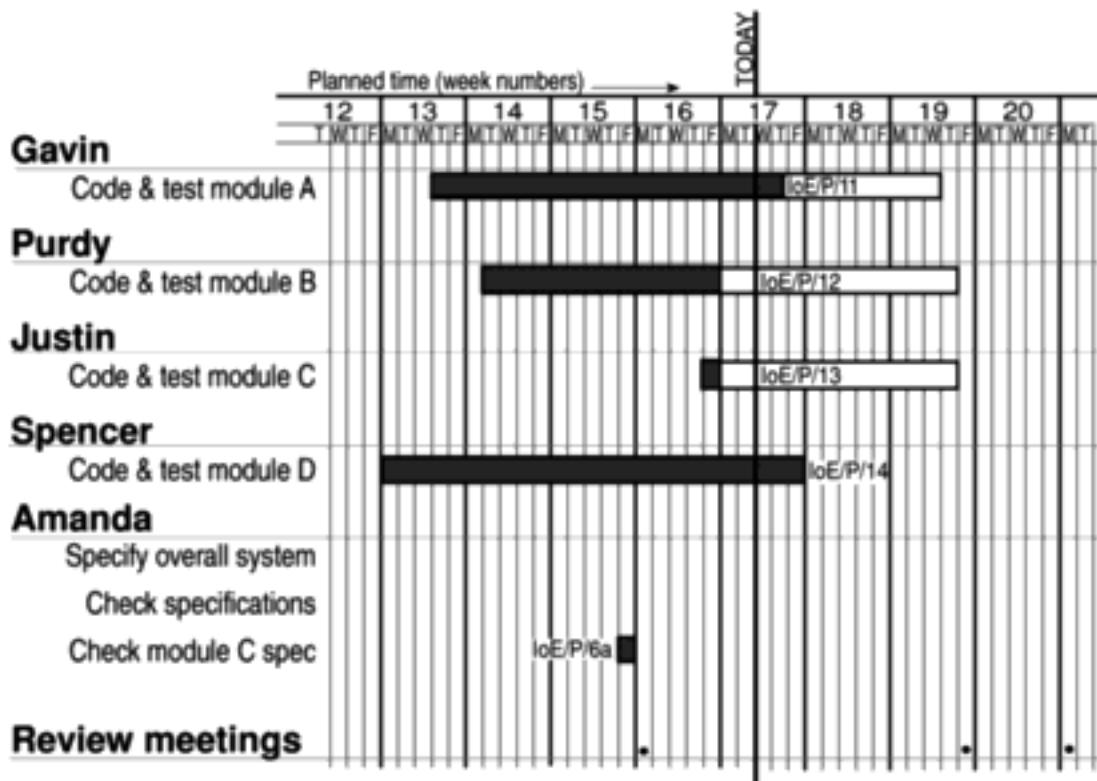


Figure 9.5 Part of Amanda's Gantt chart with the 'today cursor' in week 17.

The slip chart

A slip chart (Figure 9.6) is a very similar alternative favoured by some project managers who believe it provides a more striking visual indication of those activities that are not progressing to schedule – the more the slip line bends, the greater the variation from the plan. Additional slip lines are added at intervals and, as they build up, the project manager will gain an idea as to whether the project is improving (subsequent slip lines bend less) or not. A very jagged slip line indicates a need for rescheduling.

Ball charts

A somewhat more striking way of showing whether or not targets have been met is to use a ball chart as in Figure 9.7. In this version of the ball chart, the circles indicate start and completion points for activities. The circles initially contain the original scheduled dates. Whenever revisions are produced these are added as second dates in the appropriate circle until an activity is actually started or

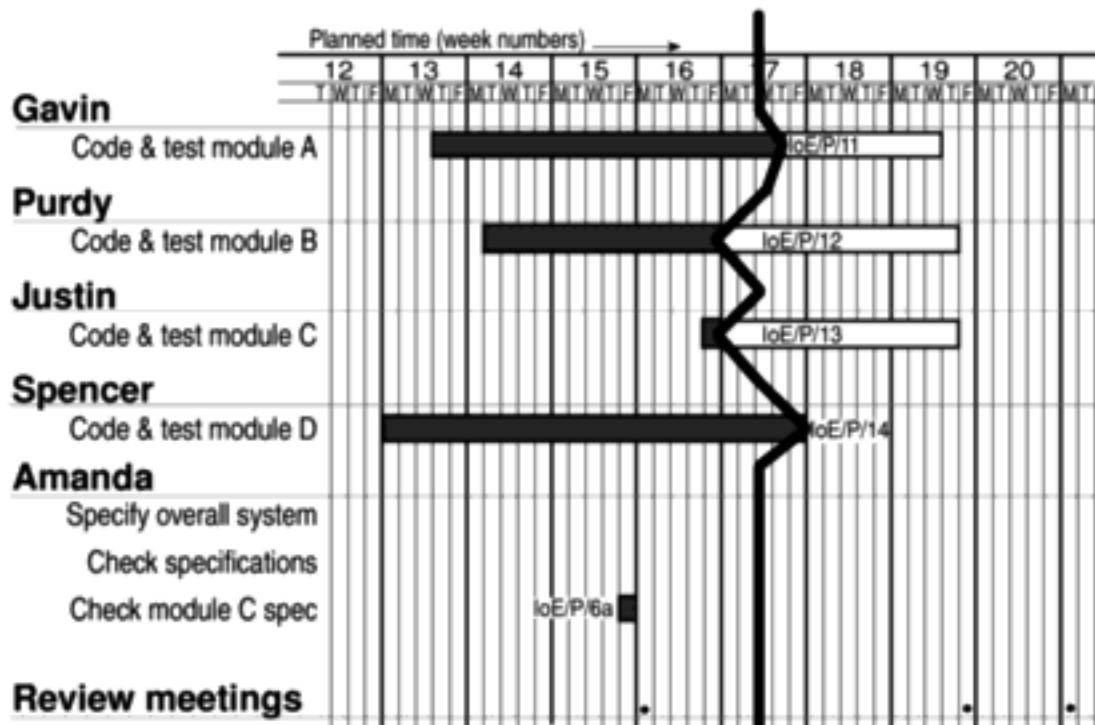


Figure 9.6 The slip chart emphasizes the relative position of each activity.

completed when the relevant date replaces the revised estimate (in bold italic in Figure 9.7). Circles will therefore contain only two dates, the original and most recent target dates, or the original and actual dates.

Where the actual start or finish date for an activity is later than the target date, the circle is coloured red (dark grey in Figure 9.7) – where an actual date is on time or earlier than the target then the circle is coloured green (light grey in Figure 9.7).

Such charts are frequently placed in a prominent position and the colour coded balls provide a constant reminder to the project team. Where more than one team is working in close proximity, such a highly visible record of achievement can encourage competitiveness between teams.

Another advantage of ball charts over Gantt and slip charts is that they are relatively easy to keep up to date – only the dates and possibly colours need to be changed, whereas the others need to be redrawn each time target dates are revised.

David Youll in *Making Software Development Visible*, John Wiley & Sons, 1990, describes a version of the ball chart using three sets of dates and part-coloured balls.

The timeline

One disadvantage of the charts described so far is that they do not show clearly the slippage of the project completion date through the life of the project. Knowing the current state of a project helps in revising plans to bring it back on target, but analysing and understanding trends helps to avoid slippage in future projects.

The timeline chart is a method of recording and displaying the way in which targets have changed throughout the duration of the project.

Figure 9.8 shows a timeline chart for Brigette's project at the end of the sixth week. Planned time is plotted along the horizontal axis and elapsed time down the vertical axis. The lines meandering down the chart represent scheduled activity

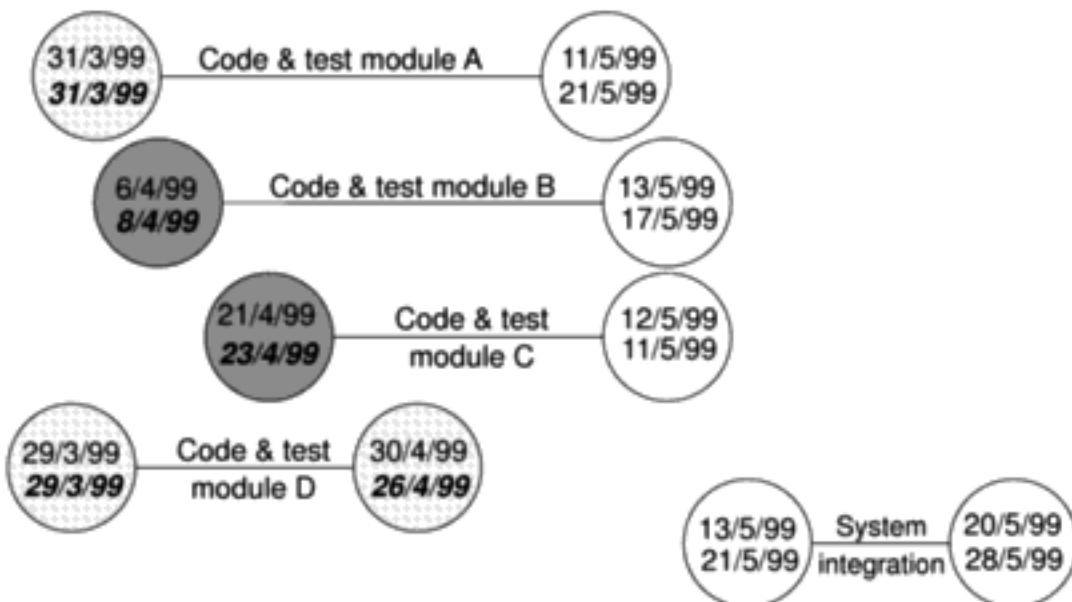


Figure 9.7 The ball wall chart provides an incentive for meeting targets.

completion dates – at the start of the project *analyse existing system* is scheduled to be completed by the Tuesday of week 3, *obtain user requirements* by Thursday of week 5, *issue tender*, the final activity, by Tuesday of week 9, and so on.

At the end of the first week Brigette reviews these target dates and leaves them as they are – lines are therefore drawn vertically downwards from the target dates to the end of week one on the actual time axis.

At the end of week two, Brigette decides that *obtain user requirements* will not be completed until Tuesday of week six – she therefore extends that activity line diagonally to reflect this. The other activity completion targets are also delayed correspondingly.

By the Tuesday of week three, *analyse existing system* is completed and Brigette puts a blob on the diagonal timeline to indicate that this has happened. At the end of week three she decides to keep to the existing targets.

At the end of week four she adds another three days to *draft tender* and *issue tender*.

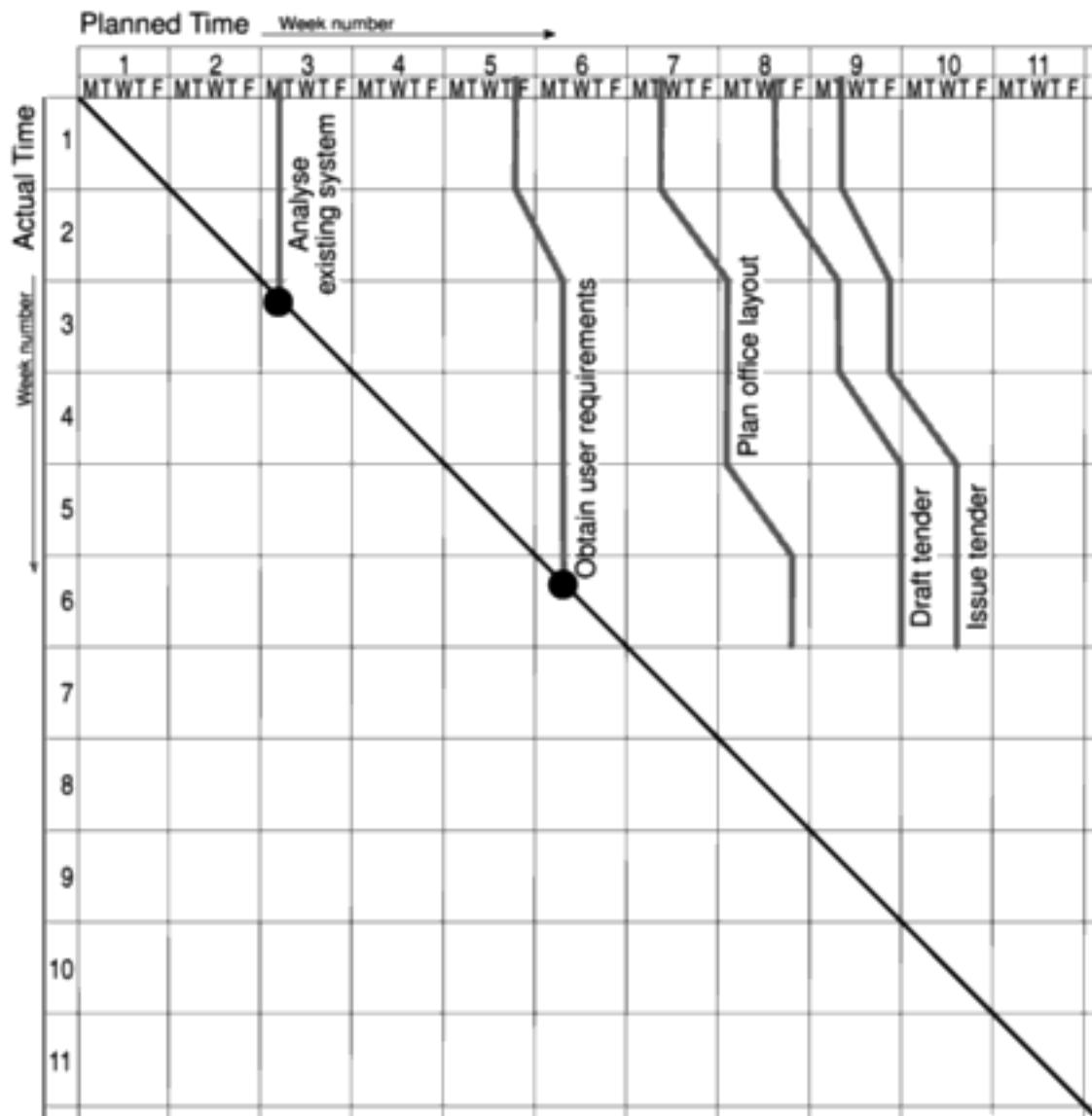
Note that, by the end of week six, two activities have been completed and three are still unfinished. Up to this point she has revised target dates on three occasions and the project as a whole is running seven days late.

Exercise 9.2

By the end of week 8 Brigette has completed planning the office layout but finds that drafting the tender is going to take one week longer than originally anticipated.

What will Brigette's timeline chart look like at the end of week 8?

If the rest of the project goes according to plan, what will Brigette's timeline chart look like when the project is completed?



Brigette's timeline chart contains only the critical activities for her project; ● indicates actual completion of an activity.

For the sake of clarity, the number of activities on a timeline chart must be limited. Using colour helps to distinguish activities, particularly where lines cross.

Figure 9.8 *Brigette's timeline chart at the end of week six.*

The timeline chart is useful both during the execution of a project and as part of the post-implementation review. Analysis of the timeline chart, and the reasons for the changes, can indicate failures in the estimation process or other errors that might, with that knowledge, be avoided in future.

9.5 Cost monitoring

Expenditure monitoring is an important component of project control. Not only in itself, but also because it provides an indication of the effort that has gone into (or at least been charged to) a project. A project might be on time but only because more money has been spent on activities than originally budgeted. A cumulative expenditure chart such as that shown in Figure 9.9 provides a simple method of comparing actual and planned expenditure. By itself it is not particularly

meaningful – Figure 9.9 could, for example, illustrate a project that is running late or one that is on time but has shown substantial costs savings! We need to take account of the current status of the project activities before attempting to interpret the meaning of recorded expenditure.

Project costs may be monitored by a company's accounting system. By themselves, they provide little information about project status.

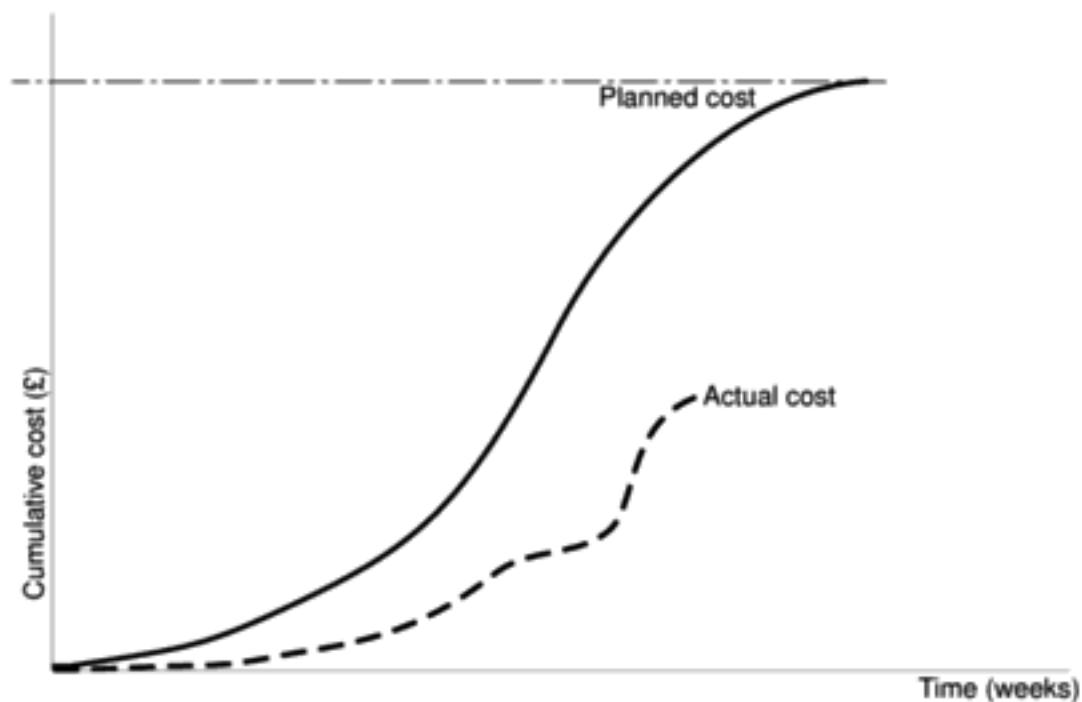


Figure 9.9 Tracking cumulative expenditure.

Cost charts become much more useful if we add projected future costs calculated by adding the estimated costs of uncompleted work to the costs already incurred. Where a computer-based planning tool is used, revision of cost schedules is generally provided automatically once actual expenditure has been recorded. Figure 9.10 illustrates the additional information available once the revised cost schedule is included – in this case it is apparent that the project is behind schedule and over budget.

9.6 Earned Value

Earned Value Analysis, also known as Budgeted Cost of Work Performed, is recommended by a number of agencies including the US and Australian departments of defence. It is also recommended in BS 6079.

Earned Value Analysis has gained in popularity in recent years and may be seen as a refinement of the cost monitoring discussed in the previous section. Earned Value Analysis is based on assigning a 'value' to each task or work package (as identified in the WBS) based on the original expenditure forecasts. The assigned value is the original budgeted cost for the item and is known as the *baseline budget* or *budgeted cost of work scheduled* (BCWS). A task that has not started is assigned the value zero and when it has been completed, it, and hence the project, is credited with the value of the task. The total value credited to a project at any point is known as the *earned value* or *budgeted cost of work performed* (BCWP) and this can be represented as a value or as a percentage of the BCWS.

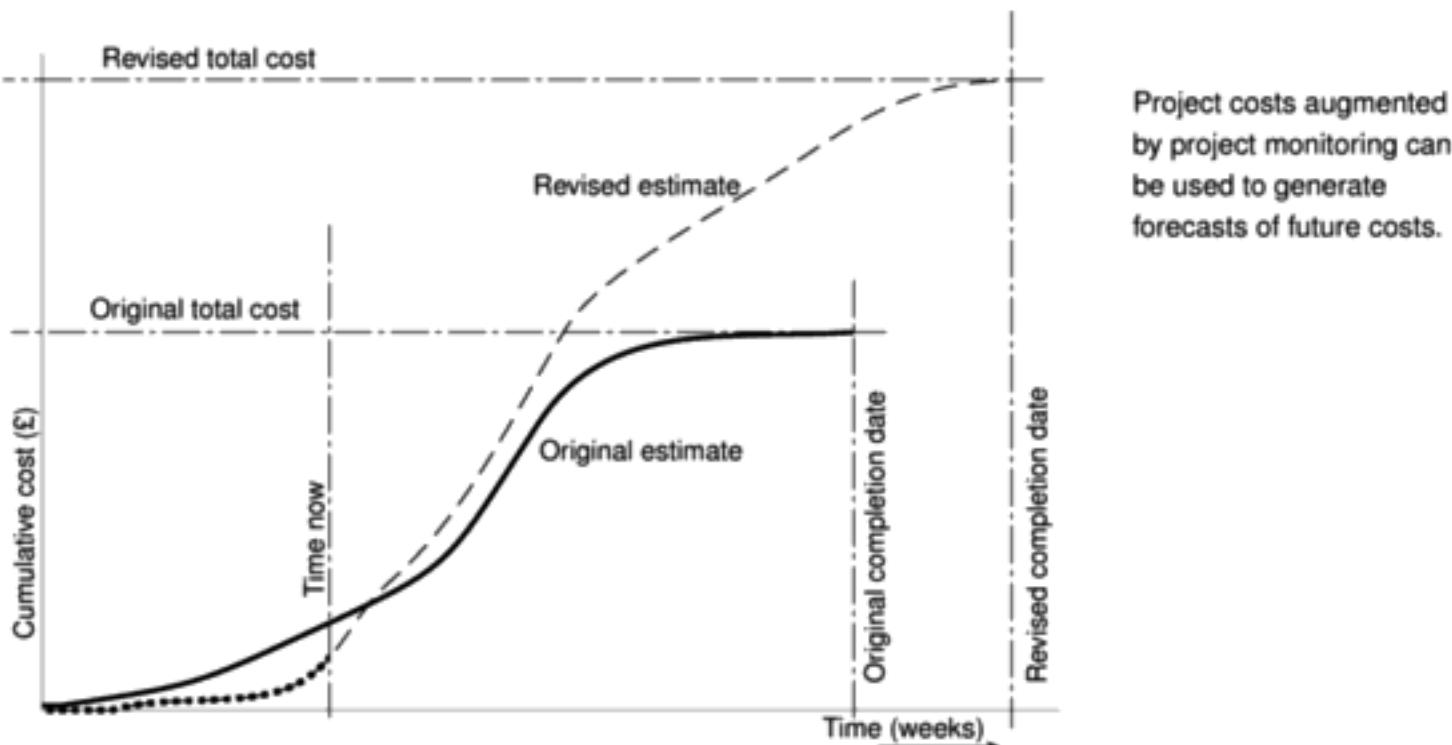


Figure 9.10 The cumulative expenditure chart can also show revised estimates of cost and completion date.

Where tasks have been started but are not yet complete, some consistent method of assigning an earned value must be applied. Common methods in software projects are:

- **the 0/100 technique** Where a task is assigned a value of zero until such time that it is completed when it is given a value of 100% of the budgeted value;
- **the 50/50 technique** Where a task is assigned a value of 50% of its value as soon as it is started and then given a value of 100% once it is complete;
- **the milestone technique** Where a task is given a value based on the achievement of milestones that have been assigned values as part of the original budget plan.

Of these, we prefer the 0/100 technique. The 50/50 technique can give a false sense of security by over-valuing the reporting of activity starts. The milestone technique might be appropriate for activities with a long duration estimate but, in such cases, it is better to break that activity into a number of smaller ones.

The baseline budget

The first stage in setting up an earned value analysis is to create the *baseline budget*. The baseline budget is based on the project plan and shows the forecast growth in earned value through time. Earned value may be measured in monetary values but, in the case of staff-intensive projects such as software development, it

is common to measure earned value in person-hours or workdays. Amanda's baseline budget, based on the schedule shown in Figure 8.7, is shown in Table 9.2 and diagrammatically in Figure 9.11. Notice that she has based her baseline budget on workdays and is using the 0/100 technique for crediting earned value to the project.

Table 9.2 Amanda's baseline budget calculation

Task	Budgeted workdays	Scheduled completion	Cumulative workdays	% cumulative earned value
Specify overall system	34	34	34	14.35
Specify module B	15	49		
Specify module D	15	49	}	27.00
Specify module A	20	54	84	35.44
Check specifications	2	56	86	36.28
Design module D	4	60	90	37.97
Design module A	7	63	97	40.93
Design module B	6	66	103	43.46
Check module C spec	1	70	104	43.88
Specify module C	25	74	129	54.43
Design module C	4	79	133	56.12
Code & test module D	25	85	158	66.67
Code & test module A	30	93	188	79.32
Code & test module B	28	94		
Code & test module C	15	94	}	97.47
System integration	6	100	237	100.00

Amanda's project is not expected to be credited with any earned value until day 34, when the activity *specify overall system* is to be completed. This activity was forecast to consume 34 person-days and it will therefore be credited with 34 person-days earned value when it has been completed. The other steps in the baseline budget chart coincide with the scheduled completion dates of other activities.

Monitoring earned value

Having created the baseline budget, the next task is to monitor earned value as the project progresses. This is done by monitoring the completion of tasks (or activity starts and milestone achievements in the case of the other crediting techniques).

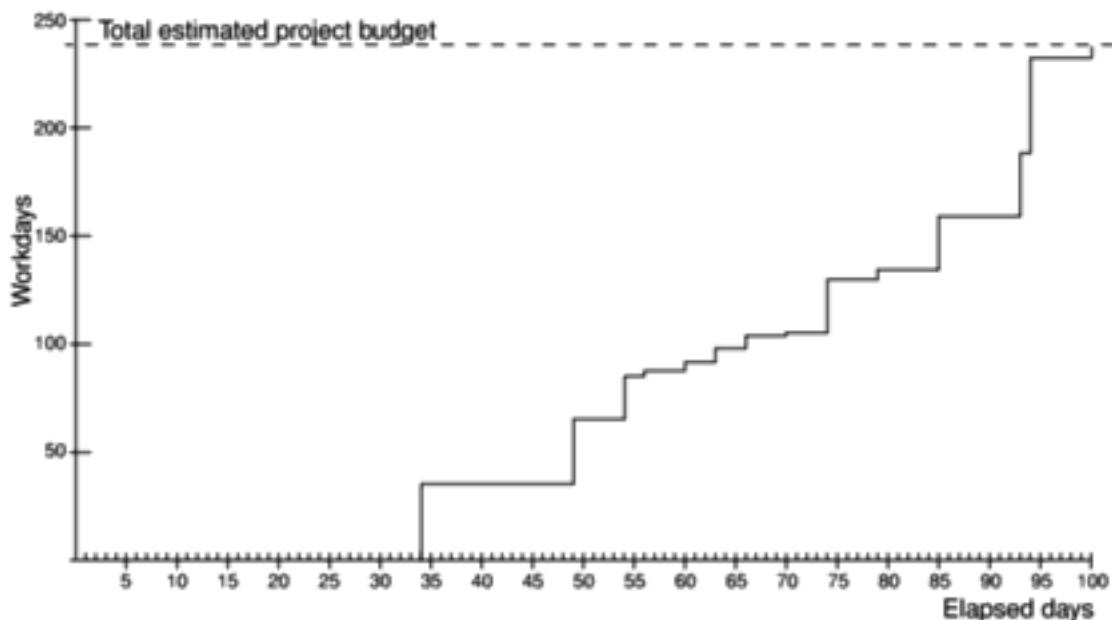


Figure 9.11 Amanda's baseline budget.

Figure 9.12 shows Amanda's earned value analysis at the start of week 12 of the project. The earned value (BCWP) is clearly lagging behind the baseline budget, indicating that the project is behind schedule.

By studying Figure 9.12, can you tell exactly what has gone wrong with her project and what the consequences might be?

Exercise 9.3

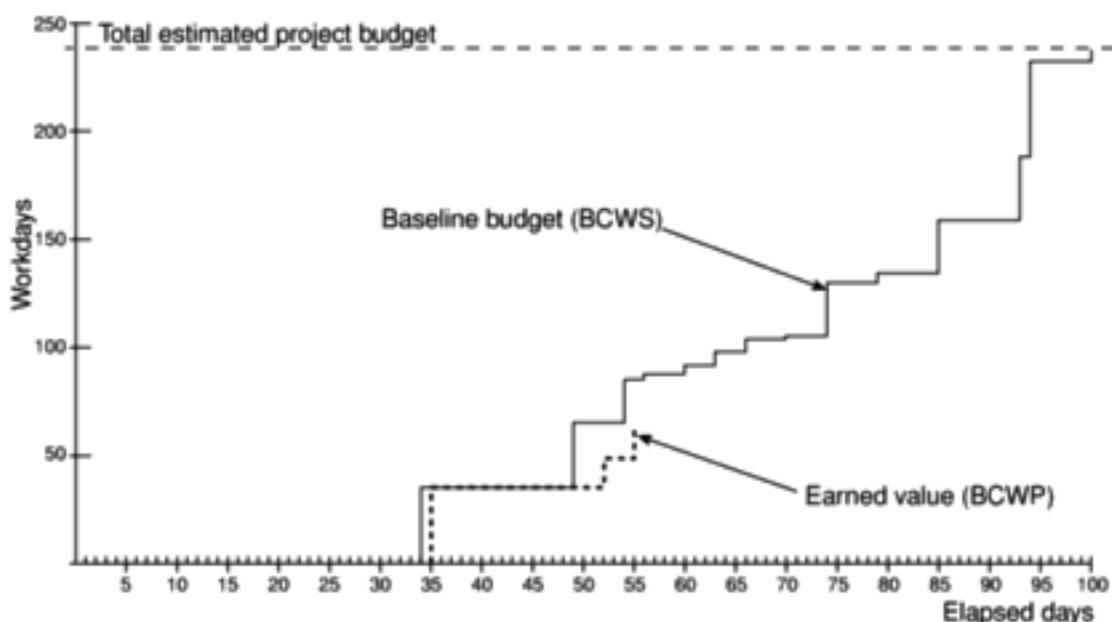


Figure 9.12 Amanda's earned value analysis at week 12.

As well as recording BCWP, the actual cost of each task can be collected as *actual cost of work performed*, ACWP. This is shown in Figure 9.13, which, in this case, records the values as percentages of the total budgeted cost.

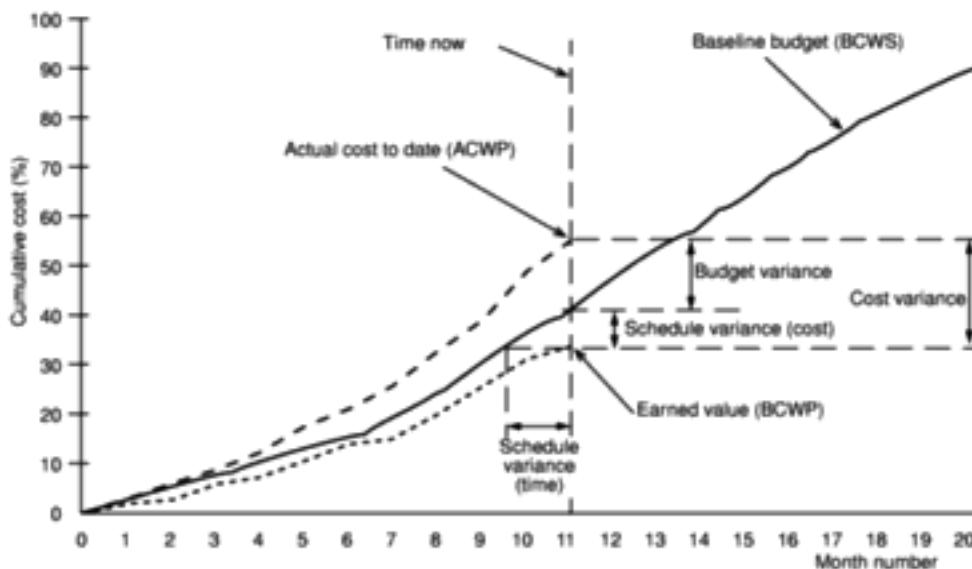


Figure 9.13 An earned value tracking chart.

Figure 9.13 also illustrates the following performance statistics, which can be shown directly or derived from the earned value chart.

Budget variance This can be calculated as $ACWP - BCWS$ and indicates the degree to which actual costs differ from those planned.

Schedule variance The schedule variance is measured in cost terms as $BCWP - BCWS$ and indicates the degree to which the value of completed work differs from that planned. Figure 9.13 also indicates the schedule variance in time, which indicates the degree to which the project is behind schedule.

Cost variance This is calculated as $BCWP - ACWP$ and indicates the difference between the budgeted cost and the actual cost of completed work. It is also an indicator of the accuracy of the original cost estimates.

Performance ratios Two ratios are commonly tracked: the *cost performance index* ($CPI = BCWP/ACWP$) and the *schedule performance index* ($SPI = BCWP/BCWS$). They can be thought of as a ‘value-for-money’ indices. A value greater than one indicates that work is being completed better than planned whereas a value of less than one means that work is costing more than and/or proceeding more slowly than planned.

In the same way that the expenditure analysis in Figure 9.9 was augmented to show revised expenditure forecasts, we can augment the simple Earned Value tracking chart with forecasts as illustrated in Figure 9.14.

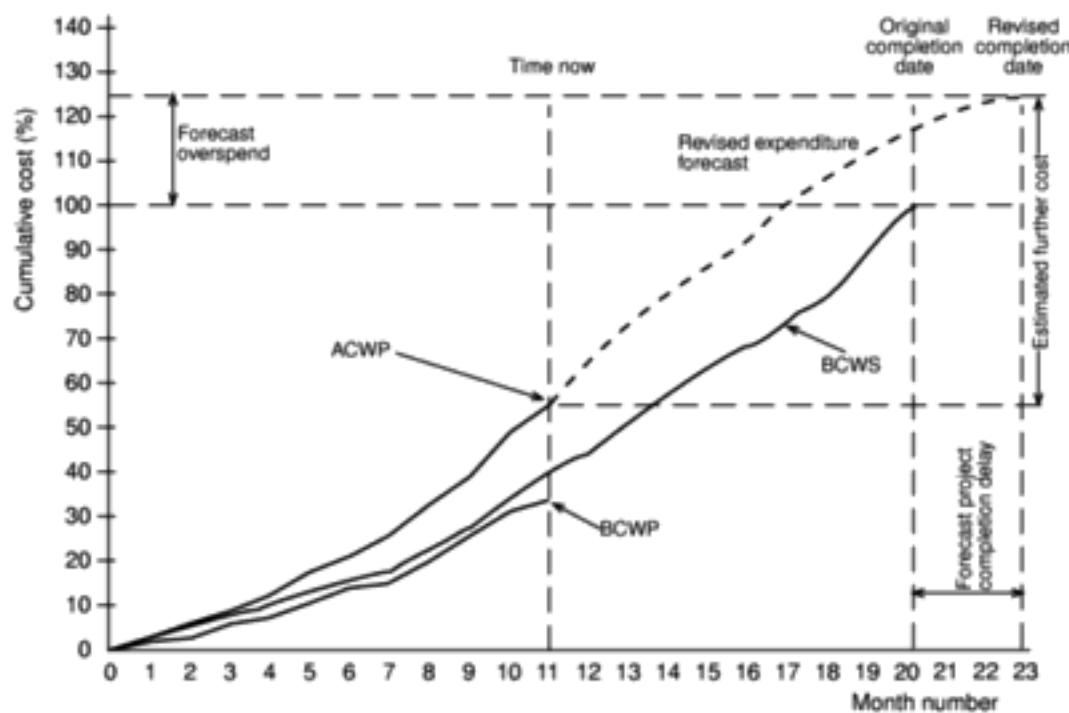


Figure 9.14 An *Earned Value* chart with revised forecasts.

Earned value analysis has not yet gained universal acceptance for use with software development projects, perhaps largely because of the attitude that, whereas a half-built house has a value reflected by the labour and materials that have been used, a half-completed software project has virtually no value at all. This is to misunderstand the purpose of earned value analysis, which, as we have seen, is a method for tracking what has been achieved on a project – measured in terms of the budgeted costs of completed tasks or products.

9.7 Prioritizing monitoring

So far we have assumed that all aspects of a project will receive equal treatment in terms of the degree of monitoring applied. We must not forget, however, that monitoring takes time and uses resources that might sometimes be put to better use!

In this section we list the priorities we might apply in deciding levels of monitoring.

- **Critical path activities** Any delay in an activity on the critical path will cause a delay in the completion date for the project. Critical path activities are therefore likely to have a very high priority for close monitoring.
- **Activities with no free float** A delay in any activity with no free float will delay at least some subsequent activities even though, if the delay is less than the total float, it might not delay the project completion date. These subsequent delays can have serious effects on our resource schedule as a delay in a

Free float is the amount of time an activity may be delayed without affecting any subsequent activity.

subsequent activity could mean that the resources for that activity will become unavailable before that activity is completed because they are committed elsewhere.

- **Activities with less than a specified float** If any activity has very little float it might use up this float before the regular activity monitoring brings the problem to the project manager's attention. It is common practice to monitor closely those activities with less than, say, one week free float.
- **High risk activities** A set of high risk activities should have been identified as part of the initial risk profiling exercise. If we are using the PERT three-estimate approach we will designate as high risk those activities that have a high estimated duration variance. These activities will be given close attention because they are most likely to overrun or overspend.
- **Activities using critical resources** Activities can be critical because they are very expensive (as in the case of specialized contract programmers). Staff or other resources might be available only for a limited period, especially if they are controlled outside the project team. In any event, an activity that demands a critical resource requires a high level of monitoring.

9.8 Getting the project back to target

A contingency plan should, of course, already exist as a result of the risk analysis methods described in Chapter 7.

The schedule is not sacrosanct – it is a plan that should be adhered to so long as it is relevant and cost-effective.

Almost any project will, at one time or another, be subject to delays and unexpected events. One of the tasks of the project manager is to recognize when this is happening (or, if possible, about to happen) and, with the minimum delay and disruption to the project team, attempt to mitigate the effects of the problem. In most cases, the project manager tries to ensure that the scheduled project end date remains unaffected. This can be done by shortening remaining activity durations or shortening the overall duration of the remaining project in the ways described in the next section.

It should be remembered, however, that this might not always be the most appropriate response to disruptions to a plan. There is little point in spending considerable sums in overtime payments in order to speed up a project if the customer is not overly concerned with the delivery date and there is no other valuable work for the team members once this project is completed.

There are two main strategies to consider when drawing up plans to bring a project back on target – shortening the critical path or altering the activity precedence requirements.

Shorten the critical path

The overall duration of a project is determined by the current critical path, so speeding up non-critical path activities will not bring forward a project completion date.

Extolling staff to 'work harder' might have some effect, although frequently a more positive form of action is required, such as increasing the resources available for some critical activity. Fact-finding, for example, might be speeded up by allocating an additional analyst to interviewing users. It is unlikely, however, that the coding of a small module would be shortened by allocating an additional programmer – indeed, it might be counterproductive because of the additional time needed organizing and allocating tasks and communicating.

Resource levels can be increased by making them available for longer. Thus, staff might be asked to work overtime for the duration of an activity and computing resources might be made available at times (such as evenings and week-ends) when they might otherwise be inaccessible.

Where these do not provide a sufficient solution, the project manager might consider allocating more efficient resources to activities on the critical path or swapping resources between critical and non-critical activities. This will be particularly appropriate with staff – an experienced programmer should be significantly more productive than a more junior member of the team.

By such means we can attempt to shorten the timescale for critical activities until such time as either we have brought the project back to schedule or further efforts prove unproductive or not cost-effective. Remember, however, that shortening a critical path often causes some other path, or paths, to become critical (see Section 6.16).

Time/cost trade-off: there is a general rule that timescales can be shortened by buying more (or more expensive) resources; sometimes this is true.

Reconsider the precedence requirements

If attempting to shorten critical activities proves insufficient, the next step is to consider the constraints by which some activities have to be deferred pending completion of others. The original project network would most probably have been drawn up assuming 'ideal' conditions and 'normal' working practices. It might be that, to avoid the project delivering late, it is now worth questioning whether as yet unstarted activities really do have to await the completion of others. It might, in a particular organization, be 'normal' to complete system testing before commencing user training. In order to avoid late completion of a project it might, however, be considered acceptable to alter 'normal' practice and start training earlier.

One way to overcome precedence constraints is to subdivide an activity into a component that can start immediately and one that is still constrained as before. For example, a user handbook can be drawn up in a draft form from the system specification and then be revised later to take account of subsequent changes.

If we do decide to alter the precedence requirements in such a way, it is clearly important to be aware that quality might be compromised and to make a considered decision to compromise quality where needed. It is equally important to assess the degree to which changes in work practices increase risk. It is possible, for example, to start coding a module before its design has been completed. It would normally, however, be considered foolhardy to do so since, as well as compromising quality, it would increase the risk of having to redo some of the

coding once the final design had been completed and thus delay the project even further.

9.9 Change control

So far in this chapter, we have assumed that the nature of the tasks to be carried out has not changed. A project leader like Amanda or Brigitte might find, however, that requirements are modified because of changing circumstances or because the users get a clearer idea of what is really needed. The payroll system that Brigitte is implementing might, for instance, need to be adjusted if the staffing structure at the college is reorganized.

Other, internal, changes will crop up. Amanda might find that there are inconsistencies in the program specifications that become apparent only when the programs are coded, and these would result in amendments to the specifications.

Careful control of these changes is needed because an alteration in one document often implies changes to other documents and the system products based on that document. The Product Flow Diagrams that have been explained in Chapter 2 indicate relationships between the products of a project where this is the case.

Exercise 9.4

A change in a program specification will normally be carried through into changes to the program design and then changed code. What other products might need to be modified?

Configuration librarian's role

Control of changes and documentation ought to be the responsibility of someone who may variously be named the Configuration Librarian, the Configuration Manager or Project Librarian. Among this person's duties would be:

- the identification of all items that are subject to change control;
- the establishment and maintenance of a central repository of the master copies of all project documentation and software products;
- the setting up and running of a formal set of procedures to deal with changes;
- the maintenance of records of who has access to which library items and the status of each library item (e.g. whether under development, under test or released).

It will be recalled that it was suggested that the setting up of change control procedures might be one of the first things the Brigitte might want to do at Brightmouth College.

BS EN ISO 9001:1994
(formerly BS 5750)
requires that a formal
change control procedure
be in place.

Change control procedures

A simple change control procedure for operational systems might have the following steps.

1. One or more users might perceive a need for a modification to a system and ask for a change request to be passed to the development staff.
2. The user management consider the change request and if they approve it pass it to the development management.
3. The development management delegate a member of staff to look at the request and to report on the practicality and cost of carrying out the change. They would, as part of this, assess the products that would be affected by the change.
4. The development management report back to the user management on the findings and the user management decide whether, in view of the cost quoted, they wish to go ahead.
5. One or more developers are authorized to take copies of the master products that are to be modified.
6. The copies are modified. In the case of software components this would involve modifying the code and recompiling and testing it.
7. When the development of new versions of the product has been completed the user management will be notified and copies of the software will be released for user acceptance testing.
8. When the user is satisfied that the products are adequate they will authorize their operational release. The master copies of configuration items will be replaced.

The above steps relate to changes to operational systems. How could they be modified to deal with systems under development?

Exercise 9.5

Changes in scope of a system

A common occurrence with IS development projects is for the size of the system gradually to increase. One cause of this is changes to requirements that are requested by users.

This is sometimes called scope creep.

Think of other reasons why there is a tendency for scope creep.

Exercise 9.6

The scope of a project needs to be carefully monitored and controlled. One way is to re-estimate the system size in terms of SLOC or function points at key milestones.

9.10 Conclusions

In this chapter we have discussed the requirements for the continual monitoring of projects and the need for making progress visible. Among the important points to emerge were:

- planning is pointless unless the execution of the plan is monitored;
- activities that are too long need to be subdivided to make them more controllable;
- ideally, progress should be measured through the delivery of project products;
- progress needs to be shown in a visually striking way, such as through ball charts, in order to communicate information effectively;
- costs need to be monitored as well as elapsed time;
- delayed projects can often be brought back on track by shortening activity times on the critical path or by relaxing some of the precedence constraints.

9.11 Further exercises

1. Take a look at Amanda's project schedule shown in Figure 8.7. Identify those activities scheduled to last more than three weeks and describe how she might monitor progress on each of them on a fortnightly or weekly basis.
2. Amanda's Gantt chart at the end of week 17 (Figure 9.5) indicates that two activities are running late. What effect might this have on the rest of the project? How might Amanda mitigate the effects of this delay?
3. Table 9.2 illustrates Amanda's earned value calculations based on workdays. Revise the table using monetary values based on the cost figures that you used in Exercise 8.5. Think carefully about how to handle the costs of Amanda as project manager and the recovered overheads and justify your decisions about how you treat them.
4. If you have access to project planning software, investigate the extent to which it offers support for earned value analysis. If it does not do so directly, investigate ways in which it would help you to generate a baseline budget (BCWS) and track the earned value (BCWP).
5. Describe a set of change control procedures that would be appropriate for Brigette to implement at Brightmouth College.

Chapter 10

Managing contracts

OBJECTIVES

When you have completed this chapter, you will be able to:

- understand the advantages and disadvantages of using goods and services brought in from outside the organization;
 - distinguish among the different types of contract;
 - follow the stages needed to negotiate an appropriate contract;
 - outline the contents of a contract for goods and services;
 - plan the evaluation of a proposal or product;
 - administer a contract from its signing until the final acceptance of project completion.
-

10.1 Introduction

In the Brightmouth College scenario, the management of the college have made a decision to obtain their software from an external supplier. Given the range of payroll software on the market and their own limited capability for developing new and reliable software, this would seem sensible. Meanwhile at IOE, Amanda has available, at least in theory, a team of software developers who are employees of IOE. However, the demand for software design and construction effort will fluctuate, rising when a new project is initiated and trailing off as it is completed. In-house developers could thus have periods of intense pressure when new projects are being developed, interspersed by periods of relative idleness. The IOE management might therefore decide that it would be more cost-effective to get an outside software house to carry out the new development while a reduced group of in-house software development staff remain busy maintaining and giving support to the users of existing systems.

It is not unusual for a major organization to spend 6 to 12 months and 40% of the total acquisition and implementation budget on package evaluation with major customer service and support applications (Demian Martinez, Decision Drivers Inc., *Computing*, 23 July 1998).

It was, for example, reported that two consortia led by Sema and EDS respectively had spent £4 million over two years bidding for a UK government project to renew the IT infrastructure in the prison service – the final job was estimated as being worth £350 million (*Computing*, 13 August, 1998).

The buying in of both goods and services, rather than 'doing it yourself', is attractive when money is available but other, less flexible, types of resource, especially staff time, are in short supply. However, there are hazards for organizations who adopt this policy. Many of these potential dangers arise from the fact that considerable staff time and attention will still be needed to manage a contracted out project successfully. Although the original motivation for contracting out might have been to reduce management effort, it is essential that customer organizations such as Brightmouth College and IOE find time to make clear their exact requirements at the beginning of the planned work, and also to ensure that the goods and services that result are in fact what are actually required.

Also, it needs hardly be said that potential suppliers are more likely to be flexible and accommodating before any contract has been signed than they will be afterwards – especially if the contract is for a fixed price. All this points to the need for as much forethought and planning with an acquisition project as with an internal development project.

In the remainder of this chapter, we will first discuss the different types of contract that can be negotiated. We will then follow through the general steps that ought to be followed when placing a contract. The issues that ought to be considered when drafting a contract are then examined. We conclude by describing some of the things that need to be done while the contract is actually being executed.

Note that the bargaining position of the customer will be much stronger if their business is going to be very valuable. If you are buying a cut-price computer game from a local store, you are unlikely to be able to negotiate variations on the supplier's standard contract of sale! (Indeed, because of the inequality of the parties in such circumstances, such sales are subject to special consumer protection laws). It is reasonable for potential suppliers to weigh up carefully the time and money they are willing to spend responding to a customer's initial request, as there is no guarantee of their obtaining the final contract.

10.2 Types of contract

The external resources required could be in the form of *services*. A simple example of this could be using temporary staff on short term contracts to carry out some project tasks. At Brightmouth College, Brigitte could use temporary staff to type into the computer system the personnel details needed to set up the payroll standing data for the new system, while at IOE a decision might be made to carry out the required system building in-house but to augment the permanent staff with contract programmers for the duration of the project. A more far-reaching use of external services would be for the contractor not only to supply the new system but to also operate it on the customer's behalf. For example, it might well be worth Brightmouth College abandoning the idea of buying a package and instead getting a payroll services agency to carry out all the payroll work on their behalf.

On the other hand, the contract could be placed for the supply of a *completed software application*.

This could be:

- a *bespoke* system, that is, a system that is created from scratch specifically for one customer;
- *off-the-shelf*, which you buy ‘as is’ – this is sometimes referred to as *shrink-wrapped* software;
- *customized off-the-shelf* (COTS) software – this is a basic core system, which is modified to meet the needs of a particular customer.

Where equipment is being supplied then, in English law, this may be regarded as a contract for the supply of *goods*. In the case of the supply of software this may be regarded as supplying a service (to write the software) or the granting of a *licence* (or permission) to use the software, which remains in the ownership of the supplier. These distinctions will have legal implications.

David Bainbridge's *Introduction to Computer Law*, Pitman, 3e, 1996 is highly recommended as a guide to the legal aspects of IT contracts.

Which of the three system options (that is, bespoke, off-the-shelf or COTS) might Amanda consider with regard to the IOE maintenance group accounts system? What factors would she need to take into account?

Exercise 10.1

Another way of classifying contracts is by the way that the payment to suppliers is calculated. We will look at:

- fixed price contracts;
- time and materials contracts;
- fixed price per delivered unit contracts

The section on ways of assessing supplier payments draws heavily on material from Paul Radford and Robyn Lawrie of Charismatek Software Metrics, Melbourne, Australia.

Fixed price contracts

As the name implies, in this situation a price is fixed when the contract is signed. The customer knows that, if there are no changes in the contract terms, this is the price to be paid on the completion of the work. In order for this to be effective, the customer’s requirement has to be known and fixed at the outset. In other words, when the contract is to construct a software system, the detailed requirements analysis must already have been carried out. Once the development is under way, the customer will not be able to change their requirements without renegotiating the price of the contract.

The advantages of this method are the following.

- **Known customer expenditure** If there are few subsequent changes to the original requirements, then the customer will have a known outlay.
- **Supplier motivation** The supplier has a motivation to manage the delivery of the system in a cost-effective manner.

The disadvantages include the following.

- **Higher prices to allow for contingency** The supplier absorbs the risk for any errors in the original estimate of product size. To reduce the impact of this risk, the supplier will add a margin when calculating the price to be quoted in a tender.
- **Difficulties in modifying requirements** The need to change the scope of the requirements sometimes becomes apparent as the system is developed – this can cause friction between the supplier and customer.
- **Upward pressure on the cost of changes** When competing against other potential suppliers, the supplier will try and quote as low a price as possible. If, once the contract is signed, further requirements are put forward, the supplier is in a strong position to demand a high price for these changes.
- **Threat to system quality** The need to meet a fixed price can mean that the quality of the software suffers.

Time and materials contracts

With this type of contract, the customer is charged at a fixed rate per unit of effort, for example, per staff-hour. At the start of the project, the supplier normally provides an estimate of the overall cost based on their current understanding of the customer's requirements, but this is not the basis for the final payment.

The advantages of this approach are the following.

- **Ease of changing requirements** Changes to requirements are dealt with easily. Where a project has a research orientation and the direction of the project changes as options are explored, then this can be an appropriate method of calculating payment.
- **Lack of price pressure** The lack of price pressure can allow better quality software to be produced.

The disadvantages of this approach are:

- **Customer liability** The customer absorbs all the risks associated with poorly defined or changing requirements.
- **Lack of incentives for supplier** The supplier has no incentive to work in a cost-effective manner or to control the scope of the system to be delivered.

Because the supplier appears to be given a blank cheque, this approach does not normally find favour with customers. However, the employment of contract development staff, in effect, involves this type of contract.

Fixed price per unit delivered contracts

This is often associated with function point (FP) counting. The size of the system to be delivered is calculated or estimated at the outset of the project. The size of

the system to be delivered might be estimated in lines of code, but FPs can be more easily and reliably derived from requirements documents. A price per unit is also quoted. The final price is then the unit price multiplied by the number of units. Table 10.1 shows a typical schedule of prices.

Table 10.1 *A schedule of charges per function point*

Function point count	Function design cost per FP	Implementation cost per FP	Total cost per FP
Up to 2,000	\$242	\$725	\$967
2,001–2,500	\$255	\$764	\$1,019
2,501–3,000	\$265	\$793	\$1,058
3,001–3,500	\$274	\$820	\$1,094
3,501–4,000	\$284	\$850	\$1,134

This Table comes from D. Garmus and D. Herron, *Measuring The Software Process*, Prentice Hall, 1996.

The company who produced this table, RDI Technologies of the USA, in fact charge a higher fee per FP for larger systems. For example, a system to be implemented contains 2,600 FPs. The overall charge would be $2,000 \times \$967$, plus $500 \times \$1,1019$, plus $100 \times \$1,058$.

One problem that has already been noted is that the scope of the system to be delivered can grow during development. The software supplier might first carry out the system design. From this design, an FP count could be derived. A charge could then be made for design work based on the figures in the 'Function design cost per FP' column. This, if the designed system was counted at 1,000 FPs, would be $1000 \times \$242 = \$242,000$. If the design were implemented, and the actual software constructed and delivered, then the additional $1000 \times \$725 = \$725,000$ would be charged. If the scope of the system grows because the users find new requirements, these new requirements would be charged at the combined rate for design and implementation. For example, if new requirements amounting to 100 extra FPs were found, then the charge for this extra work would be $\$967 \times 100 = \$96,700$.

A system to be designed and implemented is counted as comprising 3,200 FPs. What would be the total charge according to the schedule in Table 10.1?

Exercise 10.2

The advantages of this approach are as follows.

- **Customer understanding** The customer can see how the price is calculated and how it will vary with changed requirements.
- **Comparability** Pricing schedules can be compared.
- **Emerging functionality** The supplier does not bear the risk of increasing functionality.

- **Supplier efficiency** The supplier still has an incentive to deliver the required functionality in a cost-effective manner (unlike with time and materials contracts).
- **Life cycle range** The requirements do not have to be definitively specified at the outset. Thus the development contract can cover both the analysis and design stages of the project.

The disadvantages of this approach are as follows.

- **Difficulties with software size measurement** Lines of code can easily be inflated by adopting a verbose coding style. With FPs, there can be disagreements about what the FP count should really be: in some cases, FP counting rules might be seen as unfairly favouring either the supplier or customer. Users, in particular, will almost certainly not be familiar with the concept of FPs and special training might be needed for them. The solution to these problems might be to employ an independent FP counter.
- **Changing requirements** Some requested changes might affect existing transactions drastically but not add to the overall FP count. A decision has to be taken about how to deal with these changes. A change made late in the development cycle will almost certainly require more effort to implement than one made earlier.

To reduce the last difficulty, one suggestion from Australia has been to vary the charge depending on the point at which they have been requested – see Table 10.2.

The table comes from the draft Acquisition of customized software policy document, published by the Department of State Development, Victoria, 1996.

Table 10.2 Examples of additional charges for changed functionality

	Pre-acceptance testing handover	Post-acceptance testing handover
Additional FPs	100%	100%
Changed FPs	130%	150%
Deleted FPs	25%	50%

Exercise 10.3

A contract stipulates that a computer application is to be designed, constructed and delivered at a cost of \$600 per FP. After acceptance testing, the customer asks for changes to some of the functions in the system amounting to 500 FPs and some new functions which amount to 200 additional FPs. Using Table 10.2, calculate the additional charge.

In addition to the three payment methods above, there are other options and permutations of options. For instance, the implementation of an agreed specification could be at a fixed price, with provision for any additions or changes to the requirements to be charged for on a per FP basis. Another example could be where the contractor has to buy in large amounts of equipment, the price of which

might fluctuate through no fault of the contractor. In this case it is possible to negotiate a contract where the final price contains a fixed portion for labour plus an amount that depends on the actual cost of a particular component used.

It is easy to see why passing on fluctuations in equipment costs can be advantageous to the contractor. However, is there any advantage to the customer in such an arrangement?

Exercise 10.4

An underlying problem with software can once again be seen when the questions of contractual obligations and payment are considered – namely its relative invisibility and its flexibility. These mean that system size and consequently development effort tends to be very difficult to judge. If contractors are realistically to quote firm prices for work, then the tasks they are asked to undertake must be carefully constrained. For example, it would be unrealistic for a contractor to be asked to quote a single price for all the stages of a development project: how can they estimate the construction effort needed when the requirements are not yet established? For this reason it will often be necessary to negotiate a series of contracts, each covering a different part of the system development life cycle.

Another way of categorizing contracts, at least initially, is according to the approach that is used in contractor selection:

- open;
- restricted;
- negotiated.

The concept of 'IS adaptations' in Euromethod (see Appendix C) supports the idea of a series of separate contracts to implement a single system.

This categorization is based on European Union regulations.

Open tendering process

In this case, any supplier can bid to supply the goods and services. Furthermore all bids that are compliant with the original conditions laid down in the *invitation to tender* must be considered and evaluated in the same way as all the others. With a major project where there are lots of bids and the evaluation process is time-consuming, this can be an expensive way of doing things.

In recent years, there has been a global movement towards removing artificial barriers that hamper businesses in one country supplying goods and services in another. Examples of this are the drives by bodies such GATT and the European Union to ensure that national governments and public bodies do not limit the granting of contracts to their fellow nationals without good reason. One element of this is the laying down of rules about how tendering processes should be carried out. In certain circumstances, these demand that an open tendering process be adopted.

GATT stands for 'General Agreement on Trade and Tariffs'. The specific part of GATT that is relevant here is the Agreement on Government Procurement. GATT covers the EU and also several other countries including the US and Japan.

Restricted tendering process

In this case, there are bids only from suppliers who have been invited by the customer. Unlike the open tendering process, the customer may at any point reduce the number of potential suppliers being considered. This is usually the best approach to be adopted. However, it is not without risk: where the resulting contract is at a fixed price, the customer assumes responsibility for the correctness and completeness of the requirements specified to the prospective suppliers. Defects in this requirements documentation sometimes allow a successful bidder subsequently to claim for additional payments.

Negotiated procedure

There are often, however, some good reasons why the restricted tendering process might not be the most suitable in some particular sets of circumstances. Say, for example, that there is a fire that destroys part of an office, including IT equipment. The key concern here is to get replacement equipment up and running as quickly as possible and there is no time to embark on a lengthy tendering process. Another situation might be where a new software application had been successfully built and implemented by an outsider, but the customer decides to have some extensions to the system. As the original supplier has staff who have complete familiarity with the existing system, it might once again be inconvenient to approach other potential suppliers via a full tendering process.

In these cases, an approach to a single supplier might be justified. However, it takes little imagination to realise that approaching a single supplier can open the customer up to charges of favouritism and should be done only where there is a clear justification.

10.3 Stages in contract placement

We are now going to discuss the typical stages in awarding a contract.

Requirements analysis

This discussion assumes that a feasibility study has already provisionally identified the need for the intended software.

Before potential supplier can be approached, you need to have a clear set of requirements. Two points need to be emphasized here. The first is that it is easy for this step to be skimped where the user has many day-to-day pressures and not much time to think about future developments. In this situation, it can be useful to bring in an external consultant to draw up a requirements document. Even here, users and their managers need to look carefully at the resulting requirements document to ensure that it accurately reflect their needs. As David Bainbridge has pointed out: '*the lack of, or defects in, the specification are probably the heart of most disputes resulting from the acquisition of computer equipment and software*'

The requirements document might typically have sections with the headings shown in Table 10.3. This requirements document is sometimes called an operational requirement or OR.

David Bainbridge *ibid*
page 135.

Table 10.3 Main sections in a requirements document

<i>Section name</i>
1 Introduction
2 A description of any existing systems and the current environment
3 The customer's future strategy or plans
4 System requirements
• mandatory
• desirable
5 Deadlines
6 Additional information required from potential suppliers

The requirements define carefully the *functions* that need to be carried out by the new application and all the necessary *inputs* and *outputs* for these functions. The requirements should also state any *standards* with which there should be compliance, and the existing systems with which the new system needs to be compatible. As well as these functional requirements, there will also need to be operational and quality requirements concerning such matters as the required response times, reliability, usability and maintainability of the new system.

In general, the requirements document should state *needs* as accurately as possible and should avoid technical specifications of possible solutions. The onus should be placed on the potential suppliers to identify the technical solutions that they believe will meet the customer's stated needs. After all, they are the technical experts who should have access to the most up-to-date information about current technology.

Each requirement needs to be identified as being either *mandatory* or *desirable*.

- **Mandatory** If a proposal does not meet this requirement then the proposal is to be immediately rejected. No further evaluation would be required.
- **Desirable** A proposal might be deficient in this respect, but other features of the proposal could compensate for it.

For example, in the case of the Brightmouth College payroll acquisition project, Brigitte might identify as a mandatory requirement that any new system should be able to carry out all the processes previously carried out by the old system. However a desirable feature might be that the new payroll software should be able to produce accounting details of staff costs in an electronic format that can be read directly by the college's accounting computer system.

Among the other details that should be included in the requirements document to be issued to potential suppliers would be requests for any information needed to help us judge the standing of organization itself. This could include financial reports, references from past customers and the CVs of key development staff.

Chapter 12 on Software Quality discusses how aspects of quality can be measured.

One suggestion is that the weighting between product criteria and supplier criteria when selecting software ought to be 50:50 (Demian Martinez, Decision Drivers Inc., Computing 23 July 1998).

Evaluation plan

Having drawn up a list of requirements, we now need to draw up a plan of how the proposals that are submitted are to be evaluated. The situation will be slightly different if the contract is for a system that is to be specially written as opposed to an off-the-shelf application. In the latter case, it is the system itself that is being evaluated while in the former situation it is a proposal for a system.

First, a means of checking that all the mandatory requirements have been met needs to be identified. The next consideration is of how the desirable requirements can be evaluated. The problem here is weighing the value of one quality against another. The ISO 9126 standard, which is discussed in the chapter on software quality, can be used to decide that one system has more 'quality' than another, but if there is a difference in price between the two, we need to be able to estimate if the increase in quality is worth the additional price. Hence 'value for money' is often the key criterion. For example, we mentioned above an instance where the existence of an accounting link file was identified as a desirable requirement in the case of the Brightmouth College payroll acquisition project. Could a financial value be placed on this? If we were to cost clerical effort at £20 an hour and we knew that four hours of clerical effort a month went into entering staffing costs into the accounting computer system, then we could conclude that over a four year period ($\text{£}20 \text{ an hour} \times 4 \text{ hours a month} \times 48 \text{ months}$), or £3840, would be saved. If system A has this feature and costs £1000 more than system B, which does not, then this would seem to give system A an advantage. If, however, system A cost £5000 more than B then the picture would be different.

It needs to be stressed that the costs to be taken into account are those for the whole of the lifetime of the proposed system, not just the costs of acquiring the system. Also, where the relationship with the supplier is likely to be ongoing, the supplier organization needs to be assessed as well as its products.

Some of these issues were touched on in Chapter 3 on project evaluation.

Exercise 10.5

One desirable feature sought in the Brightmouth College payroll is the ability to raise staff to the next point in their salary scale automatically at the beginning of each payroll year. At present, the new scale points have to be input clerically and then be checked carefully. This takes about 20 hours of staff effort each year, which can be costed at £20 an hour. System X has this feature, but system Y does not. System X also has a feature which can automatically produce bar-charts showing payroll expenditure per department. Such a report currently has to be produced twice a year by hand and on each occasion takes about 12 hours effort to complete. With System Y, changes to department names can be carried out without any coding effort, whereas in the case of System X, the supplier would charge a minimum of £300 to do this. The college authorities estimate that there is 50% chance that this could occur during the expected four year lifetime of the system. System X costs £500 more than System Y. On the basis of this information which system appears to give better value for money?

Invitation to tender

Having produced the requirements and the evaluation plan, it is now possible to issue the invitation to tender to prospective suppliers. Essentially, this will be the requirement document with a supporting letter, which may have additional information about how responses to the invitation are to be lodged. A deadline will be specified and it is hoped that by then a number of proposals with price quotations will have been received.

In English law, for a contract to exist there must be an offer on one side that must be accepted by the other side. The invitation to tender is not an offer itself but an invitation for prospective suppliers to make an offer.

Certain new problems now emerge. The requirements that have been laid down might be capable of being satisfied in a number of different ways. The customer needs to know not only a potential supplier's price but also the way in which they intend to satisfy the requirements – this will be particularly important where the contract is to build a new system from scratch.

In some relatively straight-forward cases, it would be enough to have some post-tender clarification and negotiation to resolve issues in the supplier's proposal. With more complex projects a more elaborate approach may be needed. One way of getting the detail of the suppliers' proposals elaborated is to have a two stage tendering process.

In the first stage, technical proposals are requested from potential suppliers who do not necessarily quote any prices at this stage. Some of these proposals can be dismissed out of hand as not being able to meet the mandatory requirements. With the remaining ones, discussions may be held with representatives of the suppliers in order to clarify and validate the technical proposals. The suppliers may be asked to demonstrate certain aspects of their proposals. Where short-comings in the proposal are detected, the supplier may be given the opportunity to remedy these.

This approach is recommended by the CCTA in the United Kingdom.

The result of these discussions could be a *Memorandum of Agreement* (MoA) with each prospective supplier. This is an acceptance by the customer that the proposed solution (which might have been modified during discussions) offered by the supplier satisfactorily meets the customer's requirement.

In the second stage, tenders are invited from all the suppliers who have signed individual Memoranda of Agreement. The tender would incorporate the MoA and would be concerned with the financial terms of a potential contract.

If a design has to be produced as part of the proposal made by a supplier in response to an invitation to tender, then a difficulty would be that the supplier would have to do a considerable amount of detailed design work with only a limited prospect of being paid for it. One way of reducing this burden is for the customer to choose a small number of likely candidates who will be paid a fee to produce design proposals. These can then be compared and the final contract for construction awarded to the most attractive proposal.

The ISO 12207 has a rather different approach. Once a contract for software construction is signed, the supplier develops a design, which then has to be agreed by the customer.

Evaluation of proposals

This needs to be done in a methodical and planned manner. We have already mentioned the need to produce an evaluation plan, which will describe how each proposal will be checked to see if it meets each requirement. This reduces risks of requirements being missed and ensures that all proposals are treated consistently. Otherwise, there is a risk that a proposal might be unfairly favoured because of the presence of a feature that was not requested in the original requirement.

It will be recalled that an application could be either bespoke, off-the-shelf, or customized. In the case of off-the-shelf packages, it would be the software itself that would be evaluated and it might be possible to combine some of the evaluation with acceptance testing. With bespoke development it would be a proposal that is evaluated, while COTS may involve elements of both. Thus different planned approaches would be needed in each case.

The process of evaluation may include:

- scrutiny of the proposal documents;
- interviewing suppliers' representatives;
- demonstrations;
- site visits;
- practical tests.

The proposal documents provided by the suppliers can be scrutinized to see if they contain features satisfying all the original requirements. Clarification might need to be sought over certain points. Any factual statements made by a supplier imply a legal commitment on their part if it influences the customer to offer the contract to that supplier. It is therefore important to get a written, agreed, record of these clarifications. The customer may take the initiative here by keeping notes of meetings and then writing afterwards to the suppliers to get them to confirm the accuracy of the notes. A supplier might, in the final contract document, attempt to exclude any commitment to any representations that have been made in pre-contract negotiations – the terms of the contract need to be scrutinized in this respect.

Where the delivered product is to be based on an existing product it might be possible to see a demonstration. A danger with demonstrations is that they can be controlled by the supplier and as a passive observer it is often difficult to maintain full attention for more than, say, half-an-hour. Because of this, the customer should produce a schedule of what needs to be demonstrated, ensuring that all the important features are seen in operation.

With off-the-shelf software, it should be possible to have actual access to the application. For example, a demonstration version, which closes itself down after 30 days, might be available. Once again a test plan is needed to ensure that all the important features are evaluated in a complete and consistent manner. Once a particular package emerges as the most likely candidate, it needs to be carefully

investigated to see if there are any previously unforeseen factors that might invalidate this choice.

A frequent problem is that while an existing application works well on one platform with a certain level of transactions, it does not work satisfactorily on the platform that the customer has or at the level of throughput that it would be subjected to in the customer's work environment. Demonstrations will not generally reveal this problem. Visits to operational sites already using the system will be more informative in this respect. In the last resort a special volume test could be conducted.

How would you evaluate the following aspects of a proposal?

Exercise 10.6

- i. The usability of an existing software application.
 - ii. The usability of a software application that is yet to be designed and constructed.
 - iii. The maintenance costs of hardware to be supplied.
 - iv. The time taken to respond to requests for software support.
 - v. Training.
-

Eventually a decision will be made to award the contract to one of the suppliers. One of the central reasons for using a structured and, as far as possible, objective approach to evaluation is to be able to demonstrate that the decision has been made impartially and on merit. In most large organizations, placing a contract involves the participation of a second party within the organization, such as a contracts department, who can check that the correct procedures have been carried out. Also the final legal format of a contract will almost certainly require some legal expertise.

Where substantial sums of money are involved, legal advice on the terms of the contract is essential.

In any case, not only should the successful candidate be notified but the unsuccessful candidates should also be told of the decision. This is not simply a matter of courtesy: under GATT or EU rules, there is a legal requirement to do this in certain circumstances. It makes dealing with unsuccessful bidders easier if they can be given clear and objective reasons why their proposals did not find favour.

10.4 Typical terms of a contract

In a textbook such as this, it is not possible to describe all the necessary content of contracts for IT goods or services. It is possible, however to outline some of the major areas of concern.

Definitions

The terminology used in the contract document may need to be defined, for example, who is meant by the words 'client' and 'supplier'.

Form of agreement

For example, is it a contact of sale, a lease, or a licence? Also, can the subject of the contract, such as a licence to use a software application, be transferred to another party?

Goods and services to be supplied

Equipment and software to be supplied This includes an actual list of the individual pieces of equipment to be delivered, complete with the specific model numbers.

Services to be provided This covers such things as:

- training;
- documentation;
- installation;
- conversion of existing files;
- maintenance agreements;
- transitional insurance arrangements.

Ownership of the software

Who has ownership of the software? There are two key issues here: firstly, whether the customer can sell the software to others and, secondly, whether the supplier can sell the software to others. Where off-the-shelf software is concerned, the supplier often simply grants a license for you to use the software. Where the software is being written specially for a customer, then that customer will normally wish to ensure exclusive use of the software – they may object to software which they hoped would give them a competitive edge being sold to their rivals. They could do this by acquiring the copyright to the software outright or by specifying that they should have *exclusive use* of the software. This would need to be written into the contract. Where a core system has been customized by a supplier, then there is less scope for the customer to insist on exclusive use.

Where software is written by an employee as part of a contract of employment, it is assumed that the copyright belongs to the employer. Where the customer organization has contracted an external supplier to write software, the contract needs to make clear who is going to retain the copyright – it cannot, in this case, be automatically assumed it is the customer. The customer might have decided to take over responsibility for maintenance and further development once the software is delivered and in this case will need access to the source code. In other

cases, where the customer does not have an adequate in-house maintenance function, the supplier can retain the source code, and the customer will have to approach the supplier for any further changes. There are many potential dangers with this, not the least being that the supplier could go out of business. An escrow agreement can be included in the contract so that up-to-date copies of the source code are deposited with a third party. In the United Kingdom, the National Computing Centre provide an escrow service.

Environment

Where physical equipment is to be installed, the demarcation line between the supplier's and customer's responsibilities with regard to such matters as accommodation and electrical supply needs to be specified. Where software is being supplied, the compatibility of the software with the existing hardware and operating system platforms would need to be confirmed.

Customer commitments

Even when work is carried out by external contractors, a development project still needs the participation of the customer. The customer will have to provide accommodation for the suppliers and perhaps other facilities such as telephone lines.

Acceptance procedures

Good practice would be to accept a delivered system only after it has undergone user acceptance tests. This part of the contract would specify such details as the time that the customer will have to conduct the tests, deliverables upon which the acceptance tests depend and the procedure for signing off the testing as completed.

Some customers find that specially written or modified software is not thoroughly tested by the supplier before delivery. Some suppliers seem to think that it is cheaper to get the customer to do the testing for them!

Standards

This covers the standards with which the goods and services should comply. For example, a customer can require the supplier to conform to the ISO 12207 standard relating to the software life cycle and its documentation (or, more likely, a customized sub-set of the standard). Within the European Union, government customers with contracts for projects above a certain threshold value must, by law, ensure that the work conforms to certain standards.

Project and quality management

The arrangements for the management of the project must be agreed. Among these would be frequency and nature of progress meetings and the progress information to be supplied to the customer. The contract could require that appropriate ISO 9000-series standards be followed. The ISO 12207 standard provides for the customer to have access to quality documentation generated internally by the supplier, so that the customer can ensure that there is adherence to standards.

Timetable

This provides a schedule of when the key parts of the project should be completed. This timetable will commit both the supplier and the customer. For example, the supplier might be able to install the software on the agreed date only if the customer makes the hardware platform available at that point.

Price and payment method

Obviously the price is very important! What also needs to be agreed is when the payments are to be made. The supplier's desire to be able to meet costs as they are incurred needs to be balanced by the customer's requirement to ensure that goods and services are satisfactory before parting with their money.

Miscellaneous legal requirements

This is the legal small print. Contracts often have clauses that deal with such matters as the legal jurisdiction that will apply to the contract, what conditions would apply to the sub-contracting of the work, liability for damage to third parties, and liquidated damages. *Liquidated damages* are estimates of the financial losses that the customer would suffer if the supplier were to fall short of their obligations. It is worth noting that under English law, the penalties laid down in penalty clauses must reflect the actual losses the customer would suffer and cannot be unrealistic and merely punitive. Even this limitation will not be enough in some cases as far as the supplier is concerned. As computer systems assume increasingly critical roles in many organizations and in safety-critical systems can even be life-threatening in the case of malfunction, the possible consequential damage could be very great. Suppliers will not unnaturally try to limit this kind of liability. The courts (in England and Wales) have tended to look critically at such attempts at limiting liability, so that suppliers will, in the case of major contracts, take out insurance to cover such liabilities.

If there is a dispute, resorting to litigation, while being lucrative to the lawyers involved, is both time-consuming and expensive. An alternative is to agree that disputes be settled by *arbitration*. This requires that any dispute be referred to an expert third party whose decision as to the facts of the case is binding. Even this procedure is seldom quick and inexpensive and another option is *alternative dispute resolution* where a third party acts as a mediator who has only an advisory capacity and attempts to broker an agreement between the two sides.

10.5 Contract management

Euromethod offers guidance about how decision points can be planned.

We now need to consider the communications between the supplier and the customer while the work contracted for is being carried out. It would probably suit all concerned if the contractor could be left to get on with the work undisturbed. However, at certain *decision points*, the customer needs to examine work already done and make decisions about the future direction of the project. The project will

require representatives of the supplier and customer to interact at many points in the development cycle – for example, users need to be available to provide information needed to carry out effective detailed interface design.

This interaction, or other external factors, often leads to changes being needed, which effectively vary the terms of the contract and so a careful change control procedure is needed. Each of these topics will now be tackled in a little more detail.

When a the contract is being negotiated, certain key points in the project can be identified where customer approval is needed before the project can proceed. For example, a project to develop a large system can be divided into increments. For each increment there could be an interface design phase, and the customer needs to approve the designs before the increment is built. There could also be a decision point between increments.

Chapter 4 discusses incremental delivery.

For each decision point, the deliverables to be presented by the suppliers, the decisions to be made by the customer and the outputs from the decision point all need to be defined. These decision points have added significance if payments to the supplier are based on them. Not only the supplier but also the customer has responsibilities with respect to these decision points – for example, the supplier should not be unnecessarily delayed while awaiting customer approval of some interim deliverable.

Where work is contracted out there will be a general concern about the quality of that work. The ISO 12207 standard envisages the possibility of there being agents, employed independently of the supplier or customer, who will carry out verification, validation and quality assurance. It also allows for joint reviews of project processes and products, the nature of which needs to be clearly agreed when the contract is negotiated, otherwise the supplier might claim unwarranted interference in their work.

As the system is developed a need to change certain of the requirements often emerges. As noted earlier, essentially, this is varying the terms of the contract. Oral evidence is not normally admissible to contradict, add to, or vary the terms of a written contract, so that agreed changes need to be documented properly. An effective change control procedure is therefore needed to record requests for changes, along with the supplier's agreement to them and any fees for the additional work.

It could happen that the supplier does not meet one or more of their legal obligations. This might be through no fault of theirs, if, for example, the customer has caused the delay by being tardy in giving the necessary approvals for intermediate products. If no action is taken when the default occurs, this can be taken to imply that the customer in fact condones the failure and this can lead to the loss of a right to legal redress. The customer should therefore protect their legal rights by officially notifying the supplier as soon as possible that the failure has been recognized. It will be recalled that under English law any claim for liquidated damages should be based on actual losses. From the point where the default occurs, the customer needs to keep an accurate record of the actual losses incurred as a result of the default including any consequential losses.

10.6 Acceptance

When the work has been completed, the customer needs to take action to carry out acceptance testing. The contract might put a time limit on how long acceptance testing can take, so the customer must be organized to carry out this testing before the time limit for requesting corrections expires.

We have already noted that some software houses are rather cursory with their pre-acceptance testing: the implication seeming to be that they would rather the users spent their time on testing than they themselves. This imposition can be reduced by asking to approve the supplier's internal test plans. An associated pitfall is that once the main development work is completed, the supplier not unnaturally wants to reallocate the most productive staff to other projects. The customer can find that all their problem reports are being dealt with by relative junior members of the supplier's staff, who might not be familiar with all aspects of the delivered system.

Part or all of the payment to the supplier will depend on this acceptance testing. Sometimes part of the final payment will be retained for a period of operational running and is eventually paid over if the levels of reliability are as contracted for. There is usually a period of warranty during which the supplier should fix any errors found for no charge. The supplier might suggest a very short warranty period of say 30 days. It is in the customer's interests to negotiate a more realistic period of say at least 120 days.

10.7 Summary

Some of the key points in this chapter have been:

- the successful contracting out of work requires considerable amounts of management time;
- it is easier to gain concessions from a supplier before a contract is signed than afterwards;
- alternative proposals need to be evaluated as far as possible by comparing costs over the whole lifetime of the system rather than just the acquisition costs;
- a contract will place obligations on the customer as well as the supplier;
- contract negotiation should include reaching agreement on the management of the supplier-customer relationship during the execution of the project.

10.8 Further exercises

1. At IOE, the management are considering 'out-sourcing' the maintenance accounting system, that is, getting an outside specialist organization to take over the operation, maintenance, and support activities associated with the

- system. Write a short memorandum to management outlining the advantages and disadvantages of such a re-organization.
2. In each of the following cases discuss whether the type of application software to be adopted would be most likely to be bespoke, off-the-shelf or COTS.
 - (a) A college requires a student fees application. It is suggested that the processes required in the application are similar to those of any billing system with some requirements that are peculiar to the administration of higher education.
 - (b) A computer-based application is needed at IOE to hold personnel details of staff employed.
 - (c) A national government requires a system that calculates, records and notifies individual tax-payers about income tax charges.
 - (d) A hospital needs a knowledge-based system to diagnose the causes of eye complaints.
 3. The schedule of charges per function point shown in Table 10.1 has higher rates for larger systems. Give arguments explaining why this might be justified and also arguments against.
 4. Table 10.2 has a charge of 25% and 50% of the normal rate for deleting transactions from an application. This seems to be rather high for simply removing code. What work would be involved in deleting functionality that could justify this cost?
 5. Assume that IOE has decided on a COTS solution that will replace the whole of the existing maintenance accounting system rather than simply plugging in additional modules to deal with groups accounts. Write a memorandum that Amanda could send to IOE's legal department outlining the important provisions that a contract to supply this system should have.

Chapter 11

Managing people and organizing teams

OBJECTIVES

When you have completed this chapter, you will be able to:

- identify some of the factors that influence people's behaviour in a project environment;
 - select the most appropriate people for a project;
 - understand the role of continuing training and learning;
 - increase staff motivation;
 - improve group working;
 - use the most appropriate leadership styles;
 - understand the characteristics of the various team structures that can be employed.
-

11.1 Introduction

We are going to examine some of the problems that Amanda and Brigitte could meet when dealing with the staff who will be working for them. Where possible we will see if the findings of researchers can provide any ideas about what to do.

First, we will look at some aspects of organizational behaviour (OB) research. There will be three concerns: staff selection, staff development and, which will be dealt with in more detail, staff motivation.

We will look at how the project leader can encourage effective group working and decision making while balancing this, where needed, by purposeful leadership. The final part of the chapter looks at some of the more formal aspects of organizational structures.

The issues raised in this chapter have impacts at all stages of project planning and execution but in particular at the following points (see also Figure 11.1):

- although perhaps having little control over organizational structure, the project leader needs to be aware of its implications (Step 2);
- the scope and nature of activities can be set in a way that will enhance staff motivation (Step 4);
- many risks to project success relate to staffing (Step 6);
- the qualities of individual members of staff should be taken into account when allocating staff to activities (Step 7).

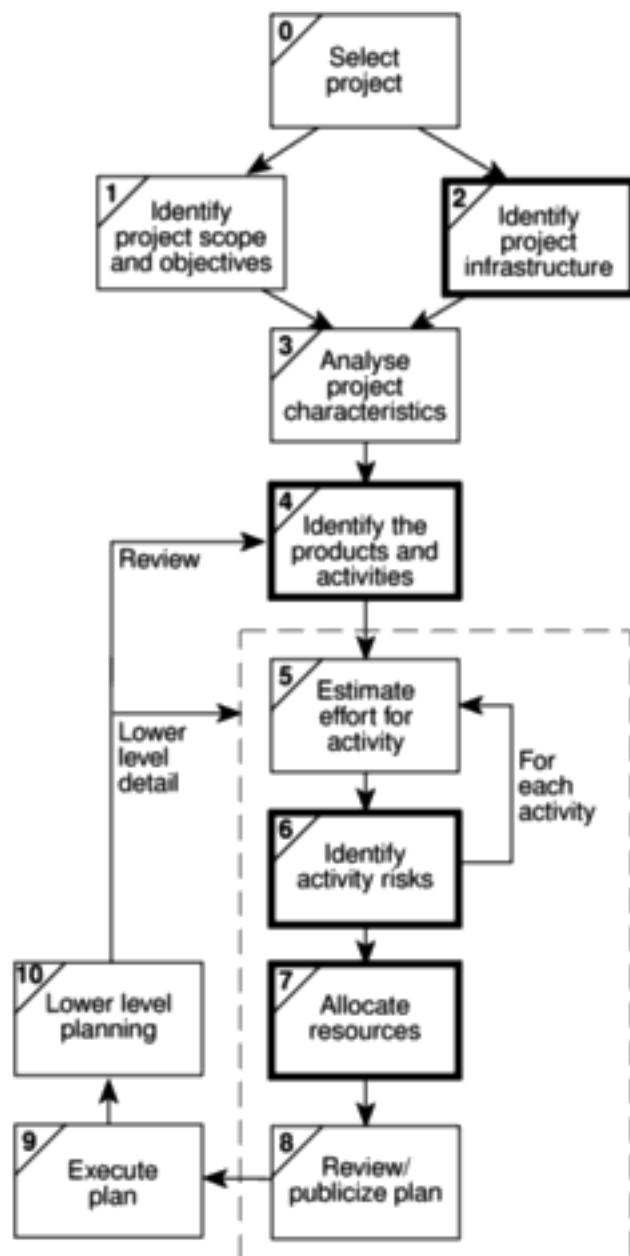


Figure 11.1 Some places in the Step Wise framework where staffing concerns are important.

11.2 Understanding behaviour

People with practical experience of working on projects invariably identify the handling of people as one of the most important aspects of project management. What people like Amanda and Brigitte will want to know is whether the effective and sensitive management of staff comes only from experience or whether guidance can be usefully sought from writers on the topic.

The field of social science known as organizational behaviour (OB) helps. This has evolved theories that try to explain people's behaviour and that tend to be structured 'If A is the situation then B is likely to result'. Attempts are then made to observe behaviour or to conduct experiments where variables for A and B are measured and a statistical relationship between the two variables is sought. Unlike physical science, it is rarely, if ever, the case that it can be said that B must always follow A.

A major problem is that in the real world there is bound to be a very wide range of influences on a situation, many of which will not be apparent to the observer. It is therefore difficult to decide which set of research findings is relevant. A danger is that we end up with a set of maxims that are little better than superstitions. However, it is hoped that by examining these questions people can become more sensitive and thoughtful about the problems involved.

In what follows, we will be making references to workers in the OB field such as Taylor, Mayo and McGregor. Rather than overwhelming the reader with references, we recommend the reader who is interested in exploring this topic further to look at Charles Handy's book. Where we have given references these tend to be for works related specifically to an IT environment.

Charles Handy,
Understanding organizations, 4th edition,
Penguin, 1993.

11.3 Organizational behaviour: a background

The roots of studies in OB can be traced back to work done in the late 19th and early 20th centuries by Frederick Taylor. By studying the way that manual workers did tasks, he attempted to work out the most productive way of doing these tasks. The workers were then trained to do the work in this way.

Taylor had three basic objectives:

- to select the best person for the job;
- to instruct such people in the best methods;
- to give incentives in the form of higher wages to the best workers.

Frederick Winslow Taylor, 1856–1915, is regarded as the father of 'scientific management' of which OB is a part.

'Taylorism' is often represented as crude and mechanistic these days. Interestingly though, the Taylorist approach is one that is adopted, in part, in modern sports coaching. A coach will attempt to get a javelin thrower, for example, to throw a javelin in a very exact manner in order to get the maximum effect. In the more mundane world of software development, the growth of

structured methods is an example of this emphasis on best practice. Both Amanda and Brigette will be concerned that tasks are carried out in the proper way. As we will see, more contentious is Taylor's emphasis on the exclusively financial basis of staff motivation, although Amanda and Brigette will be sure to find many colleagues who hold Taylor's view on the importance of 'performance-related pay'. Unfortunately, Amanda and Brigette are likely to have very little control over the financial rewards of their staff. However, they should be encouraged by findings that motivation rests not just on such rewards.

During the 1920s, OB researchers discovered, while carrying out a now famous set of tests on the conditions under which staff worked best, that not only did a group of workers for whom conditions were improved increase their work-rates, but a control group, for whom conditions were unchanged, also increased their work-rates. Simply showing a concern for what workers did increased productivity. This illustrated how the state of mind of workers influenced their productivity.

The cash-oriented view of work of some managers can thus be contrasted with a more rounded vision of people in their place of work. The two attitudes were labelled *Theory X* and *Theory Y* by Donald McGregor.

Theory X holds that:

- the average human has an innate dislike of work;
- there is a need therefore for coercion, direction and control;
- people tend to avoid responsibility.

Theory Y, on the other hand, holds that:

- work is as natural as rest or play;
- external control and coercion are not the only ways of bringing about effort directed towards the company's ends;
- commitment to objectives is a function of the rewards associated with their achievement;
- the average human can learn to accept and further seek responsibility;
- the capacity to exercise imagination and other creative qualities is widely distributed.

One way of judging whether a manager espouses Theory X or Theory Y is to observe how the manager's staff react when the boss is absent: if there is no discernible change, then this is a Theory Y environment; if everyone visibly relaxes, it is a Theory X environment. McGregor's distinction between the two theories also draws attention to the way that expectations influence behaviour. If a manager (or teacher) assumes that you are going to work diligently and do well, then you are likely to try and meet these expectations.

Elton Mayo and his colleagues did this research at the Hawthorne Works of Western Electric in Chicago, hence the 'Hawthorne Effect'.

A 'reward' does not have to be a financial reward – it could be something like a sense of achievement.

11.4 Selecting the right person for the job

Taylor stressed the need for the right person for the job. Many factors, such as the use of software tools and methodologies, affect programming productivity. However, one of the biggest differences in software development performance is among individuals. As early as 1968, a comparison of experienced professional programmers working on the same programming task found a ratio, in one case, of 1:25 between the shortest and longest time to code the program and, more significantly perhaps, of 1:28 for the time taken to debug it. Amanda and Brigitte should therefore be rightly concerned to get the best possible people working for them.

What sort of characteristics should they be looking for? Should they go, for example, for the experienced programmer or the new graduate with the first class mathematics degree? It is extremely dangerous to generalize but looking specifically at behavioural characteristics, the American researcher Cheney found that the most important influence on programmer productivity seemed to be experience. Mathematical aptitude had quite a weak influence in comparison.

Amanda and Brigitte will want staff who can communicate well with each other and, more importantly, with users. They will have some difficulties here. The American researchers Couger and Zawacki found that computing people would appear to have much weaker 'social needs' than people in other professions. They quote Gerald Weinberg: '*If asked, most programmers probably say they prefer to work alone where they wouldn't be disturbed by other people.*' This is reflected in the problem that people attracted to writing software, and are good at it, will not make good managers later in their careers.

The recruitment process

Although this is an important matter, it has to be stressed that often project leaders have little choice about the people who will make up the teams – they have to make do with the 'materials that are to hand'. Recruitment might very well be regarded as an organizational responsibility: you might be recruiting someone who will, over a period of time, work in many different parts of the same organization.

Meredith Belbin usefully distinguishes between *eligible* and *suitable* candidates. An eligible candidate is one whose CV (curriculum vitae or résumé) shows, for example, the 'right' number of years in some previous post and the 'right' paper qualifications. Suitable candidates are those who can actually do the job well. An easy mistake is to select an eligible candidate who is not in fact suitable. Suitable candidates who are not technically eligible can, on the other hand, be ideal candidates because, once in post, they are more likely to remain loyal to the organization. Belbin suggests that selection methods that centre on the assessment of actual skills rather than past experience and also a willingness to provide training to make good minor gaps in expertise can be a more effective way of placing suitable staff. It also seems to us to show that policies that avoid discrimination on the grounds of race, gender, age or irrelevant disabilities can be not just socially responsible but also a shrewd recruitment policy.

B. W. Boehm considered the quality of staff the most important influence on productivity when constructing the COCOMO software cost models (Chapter 5).

P. M. Cheney 'Effects of Individual Characteristics, Organizational Factors and Task Characteristics on Computer Programmer Productivity and Job Satisfaction' in *Information and Management*, 7 (1984).

J. D. Couger and R. A. Zawacki 'What motivates DP Professionals?' in *Datamation*, 24 (1978).

R. Meredith Belbin, *Team Roles At Work*, Butterworth-Heinemann, 1993

A general approach might be the following.

- **Create a job specification** Advice is needed, as there will be legal implications in an official document. However, formally or informally, the requirements of the job should be documented and agreed.
- **Create a job holder profile** Using the job specification, a profile of the person needed to carry out the job is constructed. The qualities, qualifications, education and experience required would be listed.
- **Obtain applicants** Typically, an advertisement would be placed, either within the organization or outside in the trade or local press. The job holder profile would be examined carefully to identify the medium most likely to reach the largest number of potential applicants at least cost. For example, if a specialist is needed it would make sense to advertise in the relevant specialist journal. The other principle is to give enough information in the advertisement to allow an element of self-elimination. By giving the salary, location, job scope and any essential qualifications, the applicants will be limited to the more realistic candidates.
- **Examine CVs** These should be read carefully and compared to the job holder profile – nothing is more annoying for all concerned than when people have CVs which clearly indicate that they are not eligible for the job and yet they are called for interview.
- **Interviews etc.** A number of different selection techniques can be tried, including aptitude tests, personality tests, and the examination of samples of previous work. All these methods must be related to specific qualities detailed in the job holder profile. Interviews are the most commonly used method. It is better if there is more than one interview session with an applicant and with each session there should not be more than two interviewers because a greater number reduces the possibility of follow-up questions and discussion. Some formal scoring system for the qualities being judged should be devised and interviewers should then decide scores individually which are then compared. An interview should be of a technical nature where the practical expertise of the candidate is assessed, or of a more general nature if not. In the latter case, a major part of the interview will in fact be evaluating and confirming what was stated in the CV – for example any time gaps in the education and employment history would be investigated, and the precise nature of jobs previously done would need to be explored.
- **Other procedures** References will need to be taken up where necessary, and a medical examination might be needed.

Exercise 11.1

A new analyst/programmer is to be recruited to work in Amanda's team at IOE. The intention is to recruit someone who already has some experience. Make a list

of the types of activities that the analyst/programmer should be capable of carrying out that can be used as the basis for a job specification.

11.5 Instruction in the best methods

This is the second concern that we have taken from Taylor. Obviously, there is a difference between loading pig iron (one of Taylor's studies) and writing C programs, but the principle of having established methods and procedures is, we hope, as well understood in software development as in steel-making.

When a new member of the team is recruited, the team leader will need to plan that person's induction into the team very carefully. Where a project is already well under way, this might not be easy. However, the effort should be made – it should pay off eventually as the new recruit will become a fully effective member of the team more quickly.

The team leader should also be aware of the need to assess continually the training needs of their team members. Just as you formulate a user requirement before considering a new system, and you construct a job holder profile before recruiting a member of staff, so a training needs profile is drawn up for each staff member before you consider specific courses. Some training can be provided by commercial training companies. Where money is tight, other sources of training should be considered but training should not be abandoned altogether even if it consists only of a team member's being told to find out about a new software tool and then demonstrating it to colleagues. Of course the nice thing about external courses is that one gets to talk to colleagues from other organizations – but attending meetings of your local branch of one of the IS/IT professional associations can serve the same purpose.

The methods learnt need, of course, to be actually applied. Reviews and inspections should help to ensure this.

The need to take into account the time needed to acclimatize new staff was stressed in Chapter 8 on resource allocation.

11.6 Motivation

The third concern that we noted from Taylor was that of motivating people to work. We are now going to look at some different models of motivation that have been proposed.

The Taylorist model

Taylor's viewpoint is reflected in the use of piece-rates in manufacturing industries and sales bonuses amongst sales forces. A problem that project leaders must be aware of is that piece-rates often cause difficulties if a new system is going to change work practices. If new technology is going to improve productivity, the question of adjusting piece-rates downwards to reflect this will be a sensitive issue.

Piece-rates are where workers are paid a fixed sum for each item they produce. Day-rates refer to payment for time worked.

Group norms are discussed further under group decision making.

Quoted by Wanda J. Orlikowski in 'Evolving with Notes: Organizational change around groupware technology' in *Groupware & Teamwork*, edited by Claudio U. Ciborra, Wiley and Sons, 1996.

Usually, radical changes in work practices have to be preceded by a move from piece-rates to day-rates.

Even where work practices are stable and output can be easily related to reward, people paid by the amount they produce will not automatically maximize their output in order to maximize their income. The amount of output will often be constrained by 'group norms', informal, even unspoken, agreements among colleagues about the amount to be produced.

Rewards have to be related in a simple and direct way to the work produced. Where a computer system is being produced, this is not easy. It is difficult to isolate and quantify work done, especially as system development and support is very much a team effort. Typical is the sentiment expressed by one member of staff in a study of software support work practices:

'This support department does well because we're a team, not because we're all individuals. I think it's the only way the support team can work successfully.'

In this kind of environment, a reward system that makes excessive distinctions between co-workers can be damaging to morale and eventually to productivity.

Exercise 11.2

A software development department wants to improve productivity by encouraging the re-use of existing software components. It has been suggested that this could be encouraged through financial rewards. To what extent do you think this could be done?

Maslow's hierarchy of needs

Different people are motivated by different things. Clearly money is a very strong motivator when you are broke. However, as the basic need for cash is satisfied, other motivators are likely to emerge. Abraham Maslow, an American psychologist, suggested that there is a hierarchy of needs. As lower levels of needs are satisfied then gradually higher level needs emerge. If these are then satisfied then yet another level of need will emerge. Basic needs are for things like food and shelter. The highest level need, according to Maslow, is the need for 'self-actualization', the feeling that you are completely fulfilling your potential.

In practice, the project leader must realise that people are likely to be motivated by different things at different stages of their life. For example, salary increases, while always welcome, probably have less of an impact on the more mature employee who is already relatively well-paid, than on a new and lowly-paid trainee. Older team-members might place more value on qualities of the job such as being allowed relative autonomy when they do their work, which shows respect for their judgment and sense of responsibility.

Exercise 11.3

Newspapers often report on the vast sums of money that are paid to the top executives of many companies. Does this mean that these people are at a low level

in the Maslow hierarchy of motivation? Do they really need all this money to be motivated? What do you think that the significance of these salaries really is?

Herzberg's two-factor theory

Certain things about a job might make you dissatisfied. If the causes of this dissatisfaction are removed, this does not necessarily make the job more exciting. On the basis of research into job satisfaction that Herzberg and his associates carried out there seemed to be two sets of factors about a job that were of importance:

- **hygiene or maintenance factors**, which can make you dissatisfied if they are not right, for example, the level of pay or the working conditions;
- **motivators**, which make you feel that the job is worthwhile, like a sense of achievement or the nature of the work itself.

Brigette, at Brightmouth College, is in an environment where it is difficult to compete with the high level of maintenance factors that can be provided by a large organization like IOE, but the smaller organization with its closer contact with the users is often able to provide better motivators.

Identify three incidents or times when you felt particularly pleased or happy about something to do with your work or study. Identify three occasions when you were particularly dissatisfied with your work or study. Compare your findings with those of your colleagues and try to identify any patterns.

Exercise 11.4

The expectancy theory of motivation

Amanda and Brigette will need to be aware of how the day-to-day ups and downs of system development affect motivation. A model of motivation developed by Vroom and his colleagues illustrates this. It identifies three influences on motivation:

- **expectancy**, the belief that working harder will lead to a better performance;
- **instrumentality**, the belief that better performance will be rewarded;
- **perceived value**, of the resulting reward.

Motivation will be high when all three factors are high. A zero level for any one of the factors can lead to a lack of motivation.

Imagine that you are trying to get a software package supplied by a third party to work. If you realize that you will never get it to work because of a bug in it, you

will give up. No matter how hard you work you will not be able to do any better (zero expectancy).

If you are working on a package for a user and, although you think you can get it to work, you discover that the user has started employing an alternative package and no longer needs this one, then you will probably feel you are wasting your time and give up (zero instrumentality).

Given that the users really do want the package, your reward in this set of circumstances might simply be a warm feeling that you have helped your colleagues and that they are grateful to you. If in fact, when the users employ the package all they do is complain and hold you responsible for any shortcomings, then you will probably avoid getting involved if they later ask for help implementing a different package (low perceived value of reward).

The Oldham–Hackman job characteristics model

Managers should try to group together the elements of the tasks that need to be carried out so that they form meaningful and satisfying assignments. Oldham and Hackman suggest that the satisfaction that a job gives is based on five factors. The first three factors make the job ‘meaningful’ to the person who is doing it:

- **skill variety**, the number of different skills that the job holder has the opportunity to exercise;
- **task identity**, the degree to which your work and its results are identifiable as belonging to you;
- **task significance**, the degree to which your job has an influence on others.

The other two factors are:

- **autonomy**, the discretion you have about the way that you do the job;
- **feedback**, the information you get back about the results of your work.

Couger and Zawacki found that programmers in general rated their jobs lower on these factors than other professions, while systems analysts and analyst-programmers rated them higher. Computer development people experienced about the same level of meaningfulness in their work as other, non-IT, professionals, but had lower perceptions of the degree of responsibility and knowledge of results of their work.

Cheney found that in the programming environment, the degree to which programmers got feedback on their work and the degree to which they could contribute to decision making had positive influences on both productivity and job satisfaction, although ‘consideration’, which was ‘the degree to which the leader develops a work climate of psychological support, mutual trust and respect, helpfulness and friendliness’, rated as less important.

In practical terms, activities should be designed so that, where possible, staff follow the progress of a particular product and feel personally associated with it.

Methods of improving motivation

- **Setting specific goals** These goals need to be demanding and yet acceptable to staff. Involving staff in the setting of goals helps to gain acceptance for them.
- **Providing feedback** Not only do goals have to be set but staff have to have regular feedback about how they are progressing.
- **Job design** Jobs can be altered to make them more interesting and give staff more feeling of responsibility.

Two measures are often used to enhance job design – job enlargement and job enrichment.

- **Job enlargement** The scope of the job is increased so that the member of staff carries out a wider range of activities. It is the opposite of increasing specialization. For example, a programmer in a maintenance group might be given responsibility for specifying minor amendments as well as carrying out the actual code changes. It is significant that Couger and Zawacki found that programmer/analysts had a higher degree of job satisfaction than programmers.
- **Job enrichment** In this case, the job is changed so that the holder carries out tasks that are normally done at a higher, managerial, level. Staff might be given responsibility for ordering consumables, for scheduling their work or for quality control. With a programmer in a maintenance team, they might be given authority to accept requests for changes which involved less than five days' work without the need for their manager's approval.

Job enlargement and job enrichment are based on the work of F. Herzberg.

11.7 Working in groups

Having discussed people as individuals, we move on to their place in groups. A key problem with major software projects is that they always involve working in groups, but as we have seen many people attracted to computer development find this difficult.

Formal groups can be subdivided into *command groups*, which are the departmental groupings that are seen on organization hierarchy diagrams and which reflect the formal management structure and *task groups* set up to deal with specific tasks. These call on people from different command groups and would typically be disbanded once the task has been completed.

11.8 Becoming a team

Simply throwing people together does not mean that they will immediately be able to work together as a team. Group feelings develop over a period of time. One suggestion is that teams go through five basic stages of development:

This classification is associated with B. W. Tuckman and M. A. Jensen.

- **forming** – the members of the group get to know each other and try to set up some ground rules about behaviour;
- **storming** – conflicts arise as various members of the group try to exert leadership and the group's methods of operation are being established;
- **norming** – conflicts are largely settled and a feeling of group identity emerges;
- **performing** – the emphasis is now on the tasks at hand;
- **adjourning** – the group disbands.

Where people are being put together into a team for the first time, then some specific team-building exercises can be undertaken. Some organizations, for example, send their management teams off on outward bound courses. Without going to these lengths, Amanda and Brigitte might try and think of some training activity which could assist in team building.

Valuable research has gone into looking at the best mix of personalities in a project team. Belbin studied teams working together on management games using various mixes of people. He initially tried putting all the people who were most able into one group. Surprisingly, these élite teams tended to do very badly – they argued a lot and as a result important tasks were often neglected.

Belbin came to the conclusion that teams needed a balance of different types of people.

- **The chair** Not necessarily a brilliant leader but must be good at running meetings, being calm, strong but tolerant.
- **The plant** Someone who is essentially very good at generating ideas and potential solutions to problems.
- **The monitor-evaluator** Good at evaluating ideas and potential solutions and helping to select the best one.
- **The shaper** Rather a worrier, who helps to direct the team's attention to the important issues.
- **The team worker** Skilled at creating a good working environment, for example by 'jollying people along'.
- **The resource investigator** Adept at finding resources in terms of both physical resources and information.
- **The completer-finisher** Good at completing tasks.
- **The company worker** A good team player who is willing to undertake less attractive tasks if they are needed for team success.

A person can have elements of more than one type. On the other hand, about 30% of the people examined by Belbin could not be classified at all! To be a good team member you must be able to:

R. Meredith Belbin
Management Teams: Why They Succeed or Fail,
 Heinemann, 1981,
 contains a self-assessment questionnaire that can help identify to which role a person is best suited.

In *Team roles at work*, 1993, Belbin suggests that 'co-ordinator' and 'implementer' are better descriptions than 'chair' and 'team worker'. A new role is added: the 'specialist', the 'techie' who likes to acquire knowledge for its own sake.

- time your interventions, that is, not overwhelm the others in the team;
- be flexible;
- be restrained;
- keep the common goals of the team in mind all the time.

Group performance

Are groups more effective than individuals working alone? Given the preference of many people attracted to software development for working on their own, this is an important question. In many projects, judgements need to be made about which tasks are best carried out collectively and which are best delegated to individuals to do on their own. As one manager at IBM was quoted as saying: '*Some work yields better results if carried out as a team while some things are slowed down if the work is compartmentalized on an individual basis*'. Part of the answer lies in the type of task being undertaken.

One way of categorizing group tasks is into:

- additive tasks;
- compensatory tasks;
- disjunctive tasks;
- conjunctive tasks.

Additive tasks are where the efforts of each participant are added together to get the final result, as in a gang of people clearing snow. The people involved are interchangeable.

With *compensatory tasks* the judgements of individual group members are pooled so that errors by some group members are compensated for by the inputs from others. An example of this would be where individual members of a group are asked to provide estimates of the effort needed to produce a piece of software and the results are then averaged. In these circumstances, group work is generally more effective than the efforts of individuals.

With *disjunctive tasks* there is only one correct answer. The effectiveness of the group depends on:

- someone coming up with the right answer;
- the others recognizing it as being correct.

With this type of task, the group can only be as good as its best member and no better.

Conjunctive tasks are where progress is governed by the rate of the slowest performer. Software production where different staff are responsible for different modules seems to be a prime example of this. The overall task is not completed until every participant's work is complete. In this case co-operative attitudes are

The IBM manager was quoted by Angelo Failla in 'Technologies for Co-ordination in a Software Factory' in *Groupware & Teamwork* edited by C. U. Ciborra, Wiley & Sons, 1996.

Code reviews could be seen as an example of a compensatory task.

The source of the quotation is the paper by Failla that is cited above.

productive: the team members who are more advanced need to ensure the meeting of group objectives by assisting those who are behind.

With all types of collective task, but particularly with additive ones, there is a danger of *social loafing*, where some individuals do not make their proper contribution. This can certainly occur with student group activities, but is not unknown in 'real' work environments. As one software developer has commented: '*[The contribution made to others] is not always recognized. Nor is the lack of any contributions ... nobody points out those who fail to make any contributions. Like when there's somebody with vital skills and you ask him for help, but he doesn't provide it*'.

Exercise 11.5

Social loafing is a problem that students often encounter when carrying out group assignments. What steps can participants in a group take to encourage team members to 'pull their weight' properly?

11.9 Decision making

Before we can look more closely at the effectiveness with which groups can make decisions we need to look in general terms at the decision-making process.

Decisions can be categorized as being:

- **structured**, generally relatively simple, routine decisions where rules can be applied in a fairly straightforward way;
- **unstructured**, more complex and often requiring a degree of creativity.

Another way of categorizing decisions is by the amount of *risk* and *uncertainty* that is involved.

Yet another distinction is between the rational-economic model and the satisficing model. The *rational-economic* model of decision making is the basis of classical economics. It predicts, for example, that a prospective buyer of personal computer equipment will purchase goods at the lowest possible price. This assumes that the decision maker has a complete knowledge of the state of the market. In order to achieve this, days, weeks, or months could be spent phoning dealers.

Sensible people probably follow a *satisficing* approach and would look at a limited number of representative outlets to get a general idea of prices. Any potential loss of money through having missed an even lower offer would probably be offset by the savings in time, phonecalls, travel and so on.

Some mental obstacles to good decision making

In this book we have rightly stressed a structured, rational, approach to decision making. Many management decisions in the real world, however, made under

Many of the techniques in Chapter 3 are attempts to make decision making more structured.

Many of the techniques in Chapter 3 on project selection are based on the rational-economic model.

Some research has found that organizations with the most comprehensive solution-seeking techniques are often the poorer financial performers!

pressure and based on incomplete information, are largely intuitive. We have to accept the role of intuition but must be aware that there are some mental obstacles to effective intuitive thinking, for example:

Faulty heuristics Heuristics mean rules of thumb. Rules of thumb can be useful but there are dangers:

- they are based only on the information that is to hand and this can be misleading;
- they are based on stereotypes, such as accepting a Welshman into a male voice choir without an audition because of the 'well-known fact' that the Welsh are a great singing nation.

Escalation of commitment This refers to the way that once you have made a decision it is increasingly difficult to alter it even in the face of evidence that it is wrong.

Information overload It is actually possible to be presented with too much information so that you 'cannot see the wood for the trees'.

Group decision making

There will be occasions where Amanda at IOE, for instance, will want to consult her whole project team about some problem. With a project team, different specialists and points of view can be brought together. Decisions made by the team as a whole are more likely to be accepted than those that are imposed upon it.

Assuming that the meetings are genuinely collectively responsible and have been properly briefed, research shows that groups are better at solving complex problems where the members of the group have complementary skills and expertise. The meeting allows them to communicate freely and to get ideas accepted.

Groups are less effective when dealing with poorly structured problems, which need creative solutions. Brainstorming techniques have been developed to help groups in this situation but research shows that people often come up with more ideas individually than in a group. Where the aim is to get the involvement of end users of a computer system, then prototyping and participatory approaches such as Joint Application Development (JAD) might be adopted.

A different type of participatory decision-making might occur when end users are consulted about the way a projected computer system is to operate.

JAD has been already discussed in Chapter 4.

Obstacles to good group decision making

Amanda finds that group decision making has some disadvantages: it is time consuming; it can in some cases stir up conflicts within the group; and decisions can be unduly influenced by dominant members of the group.

Conflict could, in fact, be less than might be expected. Experiments have shown that people will modify their personal judgements to conform to *group norms*. These are common attitudes that are developed by a group over a period of time.

You might think that this would tend to moderate the more extreme views that some individuals in the group might hold. In fact, people in groups often make

Once established group norms can survive many changes of membership in the group.

decisions that carry more risk than where they have to make the decision on their own. This is known as the *risky shift*.

Measures to reduce the disadvantages of group decision making

One method of making group decision making more efficient and effective is by training members to follow a set procedure. The *Delphi technique* endeavours to collate the judgements of a number of experts without actually bringing them face-to-face. Given a problem, the following procedure is carried out:

- the co-operation of a number of experts is enlisted;
- the problem is presented to the experts;
- the experts record their recommendations;
- these recommendations are collated and reproduced;
- the collected responses are recirculated;
- the experts comment on the ideas of others and modify their recommendations if so moved;
- if the leader detects a consensus then the process is stopped, otherwise the comments are recirculated to the experts.

The big problem with this approach used to be that because the experts could be geographically dispersed the process was time consuming.

Exercise 11.6

What developments in information technology would be of particular assistance to use of the Delphi technique?

11.10 Leadership

When Amanda and Brigette first took on project management responsibilities, one of their private anxieties was a fear that they would not have enough personal authority – that staff would not take them seriously. Leadership is generally taken to mean the ability to influence others in a group to act in a particular way in order to achieve group goals. A leader is not necessarily a good manager or vice versa, because managers have other roles to play, such as those of organizing, planning and controlling.

Authorities on this subject have found it very difficult to agree a list of the common characteristics of good leaders. It would, however, seem safe to say that they seem to have a greater need for power and achievement and have more self-control and more self-confidence than others.

Leadership is based on the idea of some kind of authority or power, although leaders do not necessarily have much formal authority. This power comes from

either the person's position (*position power*) or from the person's individual qualities (*personal power*) or can be a mixture of the two. Position power has been further analysed into:

- **coercive power**, the ability to force someone to do something by threatening punishment;
- **connection power**, which is based on having access to those who have power;
- **legitimate power**, which is based on a person's title conferring a special status;
- **reward power**, where the holder can confer rewards on those who carry out tasks to their satisfaction.

These ideas are associated with the work of J. R. P. French and B. H. Raven.

Personal power, on the other hand, can be further analysed into:

- **expert power**, which comes from being the person who is able to do a specialized task;
- **information power**, where the holder has access to information that others do not;
- **referent power**, which is based on the personal attractiveness of the leader.

What kinds of power (as defined above) would the following people have?

Exercise 11.7

- i. An internal auditor looking at the payroll system at Brightmouth College.
 - ii. A consultant who is called in to advise International Office Equipment about ways of improving software development productivity.
 - iii. The principal of Brightmouth College who has told staff that they must accept a new contract or face the sack.
 - iv. Brigette in respect to the users of the college payroll system.
 - v. Amanda in respect of the people in the project team developing the group maintenance accounts application.
-

Leadership styles

We have already suggested that Amanda and Brigette were initially concerned about establishing their personal authority. Balanced against this is the need to involve the staff in some of the decision making in order to make the best use of expertise and to gain commitment. Amanda and Brigette will need to judge when they must be authoritative and insist on things and when they must be more flexible and tolerant. Amanda, for example, might decide to be very democratic when formulating plans, but once the plans have been agreed, to insist on a very

disciplined execution of the plan. Brigitte, on the other hand, might find at Brightmouth College that she alone has the technical expertise to make some decisions, but, once she has briefed people on what needs to be done, they expect to be left alone to get on with the job as they best see fit.

Attempts have been made to measure leadership styles on two axes: directive vs. permissive and autocratic vs. democratic:

This approach is associated with Rensis Likert.

- **directive autocrat** makes decisions alone with close supervision of their implementation;
- **permissive autocrat** makes decision alone but gives subordinates latitude in implementation;
- **directive democrat** makes decisions participatively but closely supervises their implementation;
- **permissive democrat** makes decisions participatively and gives subordinates latitude in implementation.

It should be emphasized that there is no one best style of management – it depends on the situation.

Another axis on which there have been attempts to measure management qualities has been on the degree to which a manager is *task-oriented*, that is, the extent to which the execution of the task at hand is paramount, and the degree to which the manager is concerned about the people involved (*people orientation*). It is perhaps not surprising that subordinates appear to perform best with managers who score highly in both respects.

Work environments vary according to the amount of control that can be exerted over the work. Some jobs are routine and predictable (as when dealing with batched computer output). Others may be driven by outside factors (as in the case of a help-desk) or are situations where future direction is uncertain (for example, at the early stages of a feasibility study). Where there is a high degree of uncertainty, subordinates will seek guidance from above and welcome a task-oriented management style. As uncertainty is reduced, the task-oriented manager is likely to relax and to become more people-oriented and this will have good results. People-oriented managers are better where staff can control the work they do and know what to do without referring matters to their line managers. It is then argued that if control becomes even easier the people-oriented manager will be tempted to get involved in more task-centred questions and that this may have undesirable results.

Research findings also show that where team members are relatively inexperienced a task-oriented approach is most effective. As group members mature, consideration for their personal needs and aspirations becomes more valued. Where maturity is very high, then there is no need for a strong emphasis on either of these approaches.

Exercise 11.8

What in your view would be the most appropriate management style when dealing with the following subordinates?

- i. At Brightmouth College, a former member of the local authority who has dealt with the college payroll for several years and who has been employed by the college to set up and manage the new payroll section.
 - ii. At IOE, a new trainee analyst-programmer who has just joined Amanda's group.
 - iii. At IOE, a very experienced analyst-programmer aged 45, who was recruited into the software development department some time ago from the accounts department and who has been dealing with system support for the old maintenance accounts system that is now being revised.
-

11.11 Organizational structures

Formal versus informal structures

While organizational structures can have an enormous impact on the way a project is conducted, it is something that project leaders such as Amanda at IOE can often do little to change.

The *formal* structure is the one that is expressed in the staff hierarchy chart. It is basically concerned with *authority*, about who has which boss. It is backed by an *informal* structure of contacts and communication that grows up spontaneously among members of staff during the course of work. When the unexpected happens it is often this system that comes into play. Over a period of time, the advantages and disadvantages of different organizational structures tend to even out – the informal organization gets built up and staff find unofficial ways of getting around the obstacles posed by the formal structure.

Hierarchical approach

The 'traditional' management structure is based on the concept of the *hierarchy* – each member of staff has only one manager, while a manager will have responsibility for several members of staff. Authority flows from the top down through the structure. A traditional concern has been with the *span of control* – the number of people that a manager can effectively control.

Staff versus line

Staff in organizations can often be divided into *line* workers who actually produce the end product and support *staff* who carry out supporting roles. In some organizations that produce software for the market or as a component of a larger product which is sold, the software specialists might be seen as part of the line. In a financial organization, on the other hand, the information systems department would probably be seen as part of the support staff.

Departmentalization

In drawing up a structure, the question of *differentiation* crops up. This is the question of how the organization is to be departmentalized. This is often based on staff specialisms, product lines, categories of customer or geographical location, for example.

In the case of software development, it is usually the case that either a *functional* or a *task-oriented* approach is used. With functional departmentalization, systems analysts might be put in a group separate from the programmers. The programmers would act as a pool from which resources may be drawn for particular tasks. With a task-oriented approach, the programmers and systems analysts are grouped together in one project team. The project team might be gathered in order to implement a specific long-term project or might exist on a permanent basis to service the needs of a particular set of users.

One advantage of the functional approach is that it can lead to a more effective use of staff. Programmers can be allocated to jobs as needed and be released for other work when a particular task is completed. For instance, in a project team there are bound to be periods of greater and lesser coding activity and programmers might find there are spells when they are under-utilized. The functional organization will also make it easier for programmers to have careers that are technically oriented – there will probably be a career structure within the software development department that allows the programmer to rise without having to change specialism. This type of organization should also encourage the interchange of new technical ideas among technical staff and the promulgation of company wide standards.

A disadvantage is that having two separate departments can lead to communication problems, especially if a programmer is unfamiliar with the application area. There will also be problems with software maintenance – here it is helpful to have programmers who have built up a familiarity with particular parts of the application software. Users might prefer the established project team approach because, when they require new software features, they will already have a group dedicated to their needs and will not find themselves in the position of always having to fight other departments for development resources. The project team structure tends to favour a pattern of career progression where programmers eventually become systems analysts.

A third method of departmentalization is based on lifecycle phase. Here there are separate teams for development and maintenance. Some staff can concentrate in a focused and sustained manner on developing new applications with few interruptions, while other teams, more oriented towards service and support, deal with maintenance.

Some organizations have attempted to get the best of all worlds by having a *matrix* structure. In this case the programmer would have two managers: a project leader who would give day-to-day direction about the work in hand and a programming manager who would be concerned about such things as career development.

Centralized versus decentralized group structures

At the level of a project group, a decentralized organization would mean that the group members would tend to make major decisions collectively and that there would be a large degree of free communication among group members. With the centralized approach the group would be broken down into sections, each of which would be directed by a leader who communicates on behalf of the section with other groups.

Decentralized groups, because of the time taken to debate things, tend to work more slowly. They are likely to be affected by the establishment of group norms and the influence of the risky shift, which has already been described. However, they are better at dealing with complex problems while the centralized group organization deals more effectively with simple problems.

The discussion of centralized versus decentralized groups assumes that software development work has to be done as a group. In fact, given the preference of many software developers for working on their own, an organization where each programmer works in isolation can be envisaged – indeed there are software houses that are based on people working at home.

Egoless programming

In the early days of computer development, managers tended to think of the programmer as communing mysteriously with the machine. The tendency was for programmers to see programs as being an extension of themselves and to feel over-protective towards them. The effects of this on the maintainability of programs can be imagined. Gerald Weinberg made the then revolutionary suggestion that programmers and programming team leaders should read other people's programs. Programs would become in effect the common property of the programming group and programming would become 'egoless'. Peer code reviews are based on this idea. Weinberg's ideal programming team was a decentralized group freely communicating within itself.

G. M. Weinberg, *The Psychology of Computer Programming*, Van Nostrand Reibold, 1971.

Chief programmer teams

The larger the decentralized group, the slower it will get, because of the increased communication. On really large time-critical projects, a more formalized centralized structure is essential. Brooks pointed out the need for design consistency when producing a large complex system and how this might be difficult when there are a large number of people involved in producing a piece of software. One suggestion was to try to reduce the number of people actually creating software but to make these programmers as productive as possible by giving them as much support as possible.

Brooks' *Mythical Man-Month* has already been referred to. He was in charge of the huge team that created the operating system for the IBM 360 range.

The result of this train of thought was the *chief programmer* team. The chief programmer is the person who defines the specification, and designs, codes, tests and documents the software. There is also a *copilot*, with whom the chief programmer can discuss problems and who writes some code. They are supported by an *editor* to write up the documentation drafted by the chief programmer, a

program clerk to maintain the actual code, and a *tester*. The general idea is that this team is under the control of a single unifying intellect.

The chief programmer concept was used on the influential *New York Times* data bank project, where many aspects of structured programming were tried out. In this case, each chief programmer managed a senior level programmer and a program librarian. Additional members could be added to the team on a temporary basis to deal with particular problems or tasks.

The problem with this kind of organization is getting hold of really outstanding programmers to carry out the chief programmer role. There are also the dangers of information overload on the chief programmer, and of staff dissatisfaction among those who are there simply to minister to the needs of the superstar chief programmers.

Controlled decentralized groups

This compromise structure has been suggested and seems to follow common industry practice. A project team is made of groups under the leadership of senior programmers. Within these groups there is free communication and a practice of reviewing each others' work. Communication with other groups is at senior programmer level, while a project leader has overall authority.

11.12 Conclusion

Some of the important points that have been made in this chapter are:

- people may be motivated by money, but they are motivated by other things as well;
- both staff selection and the identification of training needs should be done in an orderly, structured, way where requirements are clearly defined first;
- thoughtful job design can increase staff motivation;
- consideration should be given, when forming a new project team, to getting the right mix of people and to planning activities that will promote team building;
- group working is more effective with some types of activity than others;
- different styles of leadership are needed in different situations;
- the people who need to communicate most with each other should be grouped together organizationally.

11.13 Further exercises

1. An organization has detected low job satisfaction in the following departments:
 - the system testing group;

- the computer applications help desk;
- computer batch input.

How could these jobs be redesigned to give more job satisfaction?

2. In Exercise 11.1, a job specification was requested.
 - (a) Write a job holder profile of the sort of person who would be able to fulfil the specification in terms of qualities, qualifications, previous education and experience.
 - (b) For each element in the job holder profile that you have produced in (a) above, describe ways of finding out whether an applicant has met the requirement.
3. To what extent is the Belbin approach to balanced teams compatible with having chief programmer teams?
4. If you have been involved recently in a group activity or project, try and categorize each participant according to the Belbin classification. Were there any duplications or gaps in any of the roles? Did this seem to have any impact on progress?
5. Three different mental obstacles to good decision making were identified in the text: faulty heuristics, escalation of commitment and information overload. What steps do you think can be taken to reduce the danger of each of these?
6. In Exercise 11.8, the management style most appropriate for each of three different situations was asked for. Go back and consider how you as a manager would respond to each of these three situations in terms of practical things to do or avoid.

Chapter 12

Software quality

OBJECTIVES

When you have completed this chapter you will be able to:

- explain the importance of software quality to software users and developers;
 - define the qualities of good software;
 - design methods of measuring the required qualities of software;
 - monitor the quality of the processes in a software project;
 - use external quality standards to ensure the quality of software acquired from an outside supplier;
 - develop systems using procedures that will increase their quality.
-

12.1 Introduction

While quality is generally agreed to be ‘a good thing’, in practice the quality of a system will be a vague, undefined, attribute if we are not careful. We therefore need to define precisely what qualities we require of a system. However, this by itself is not enough – we need to be able to judge objectively whether a system meets our quality requirements and this leads to the need for measurement. This would be of particular concern to someone like Brigette at Brightmouth College when she is in the process of selecting a payroll package.

For someone, like Amanda at IOE, who is developing software, waiting until the system finally exists before measuring it would be leaving things rather late. She would want to be able to forecast the likely quality of the final system while it was still under development, and also to make sure that the development methods that were used were likely to produce that quality. This leads to a slightly different emphasis – rather than concentrating on the quality of the final system, a potential customer for software might try to check that the suppliers were using the best methods.

This chapter examines these issues.

12.2 The place of software quality in project planning

Quality will be of concern at all stages of project planning and execution, but will be of particular interest at the following points in the Step Wise framework (Figure 12.1).

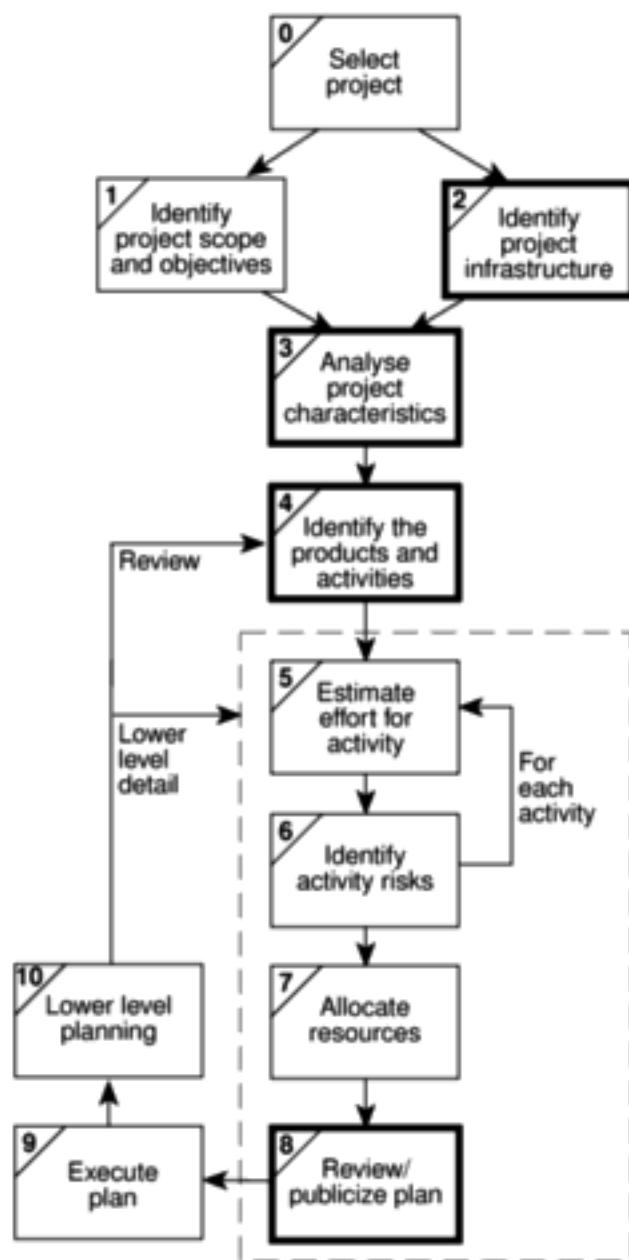


Figure 12.1 The place of software quality in Step Wise.

- **Step 2: Identify project infrastructure** Within this step, activity 2.2 identifies installation standards and procedures. Some of these will almost certainly be about quality.
- **Step 3: Analyse project characteristics** In activity 3.2 ('Analyse other project characteristics – including quality based ones') the system to be implemented will be examined to see if it has any special quality requirements.

If, for example, it is extremely safety-critical then a whole range of additional activities can be added; these include such things as *n*-version development, where a number of teams develop versions of the same software that are then run in parallel with the outputs being cross-checked for discrepancies.

- **Step 4: Identify the products and activities of the project** It is at this point that the entry, exit and process requirements are identified for each activity. The nature of these requirements is described later in this chapter.
- **Step 8: Review and publicize plan** At this stage, the overall quality aspects of the project plan are reviewed.

12.3 The importance of software quality

We would expect quality to be a concern of all producers of goods and services. However, the special characteristics of software, and in particular its intangibility and complexity, make special demands.

- **Increasing criticality of software** The final customer or user is naturally anxious about the general quality of software, especially its reliability. This is increasingly the case as organizations become more dependent on their computer systems and software is used more and more in areas that are safety-critical, for example to control aircraft.
- **The intangibility of software** This makes it difficult to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developer produce 'deliverables' that can be examined for quality.
- **Accumulating errors during software development** As computer system development is made up of a number of steps where the output from one step is the input to the next, the errors in the earlier deliverables will be added to those in the later steps leading to an accumulating detrimental effect. In general, the later in a project that an error is found the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly difficult to control.

The intangibility of software presents particular problems - see Frederick P. Brooks. 'No Silver Bullet: Essence and Accidents of Software Engineering', *IEEE Computer*, April 1987.

For these reasons quality management is an essential part of effective overall project management.

12.4 Defining software quality

Quality is a rather vague term and we need to define carefully what we mean by it. For any software system, there should be three specifications:

- a functional specification describing what the system is to do;
- a quality (or attribute) specification concerned with how well the functions are to operate;

- a resource specification concerned with how much is to be spent on the system.

Methodologies such as SSADM produce primarily functional requirements – so how are the required qualities of the proposed system, such as its flexibility when change is needed, to be defined?

Exercise 12.1

At Brightmouth College, Brigette has to select the best off-the-shelf payroll package for the college. How should she go about this in a methodical manner?

One element of the approach could well be the identification of criteria against which payroll packages are to be judged. What might these criteria be and how could you check the extent to which candidate packages measure up against these criteria?

James A. McCall, 'An Introduction to Software Quality Metrics', in: J. D. Cooper & M. J. Fisher (eds.) *Software Quality Management*, Petrocelli, 1978.

The ISO 9126 standard presents an alternative set, which is described later.

The relationship between any two quality factors can be:

- **indifferent** The presence of one quality has no effect on the other;
- **complementary** The presence of one quality would suggest the presence of the other;
- **conflicting** The presence of one quality is likely to reduce the presence of the other.

There have been several attempts to identify specific product qualities that are appropriate to software. James A. McCall, for instance, grouped software qualities into three sets:

- product operation qualities;
- product revision qualities;
- product transition qualities.

The definitions below are those given by McCall, but the reader will come across others. In particular circumstances, additional qualities are of interest.

Product operation quality factors

- **Correctness** The extent to which a program satisfies its specifications and fulfils the user's objectives.
- **Reliability** The extent to which a program can be expected to perform its intended function with required precision.
- **Efficiency** The amounts of computer resources required by the software.
- **Integrity** The extent to which access to software or data by unauthorized persons can be controlled.
- **Usability** The effort required to learn, operate, prepare input and interpret output.

Product revision quality factors

- **Maintainability** The effort required to locate and fix an error in an operational program.
- **Testability** The effort required to test a program to ensure it performs its intended function.

- **Flexibility** The effort required to modify an operational program.

Product transition quality factors

- **Portability** The effort required to transfer a program from one hardware configuration and/or software system environment to another.
- **Reusability** The extent to which a program can be used in other applications.
- **Interoperability** The effort required to couple one system to another.

Look at McCall's list of quality factors. Identify examples of pairs that are (a) indifferent, (b) complementary and (c) conflicting.

Exercise 12.2

McCall's software quality factors reflect the external view of software that users would have. For instance, usability would be a key concern of users. These quality factors have to be translated into internal factors of which the developers would be aware – *software quality criteria* (Table 12.1).

Table 12.1 *Software quality criteria*

<i>Quality factor</i>	<i>Software quality criteria</i>	
Correctness	traceability, consistency, completeness	
Reliability	error tolerance, consistency, accuracy, simplicity	
Efficiency	execution efficiency, storage efficiency	
Integrity	access control, access audit	
Usability	operability, training, communicativeness, input/output volume, input/output rate	
Maintainability	consistency, simplicity, conciseness, modularity, self-descriptiveness	The same software quality criterion can apply to more than one of the software quality factors.
Testability	simplicity, modularity, instrumentation, self-descriptiveness	
Flexibility	modularity, generality, expandability, self-descriptiveness	
Portability	modularity, self-descriptiveness, machine independence, software system independence	
Reusability	generality, modularity, software system independence, machine independence, self-descriptiveness	
Interoperability	modularity, communications commonality, data commonality	

Exercise 12.3

The same software quality criteria often appear for more than one software quality factor. What is the significance of this?

Measures can be:

- **relative quantity measures** where an attempt is made to quantify the presence of the quality, or
- **binary measures** where the quality is deemed either to be present or not present.

Some writers use the term *metric* interchangeably with *measure*. Software measurement specialists would, however, maintain that there is a technical difference between the two terms.

Defining quality is not enough. If we are to judge whether a system meets our requirements we need to be able to measure its qualities. For each criterion, one or more measures have to be invented to assess the degree to which the quality is present.

Any good relative measure must be able to relate the number of units to the maximum possible in the circumstances. The maximum number of faults in a program, for example, is going to be related to the size of the program, so a measure of 'faults per thousand lines of code' is more helpful than 'total faults in a program' as a means of judging the quality of a program.

Trying to find measures for a particular quality helps to clarify ideas about what that quality really is. What is being asked is, in effect, 'how do we know when we have been successful?' An answer to this is essential if the quality objectives are to be communicated to a large number of people.

In some cases we can measure the quality directly while in other cases the thing being measured is not the quality itself but an indicator of the degree to which the quality is present. By identifying measures, management are setting targets for project team members so care has to be taken that an improvement in the measured attribute is always going to be valid. For example, the number of errors found in program inspections could be counted. This count could, of course, be improved by allowing more errors to go through to the inspection stage rather than eradicating them earlier – which is not quite the point!

In general, the user of software would be concerned with measuring what McCall called *quality factors* while the developers would be concerned with *quality criteria*.

The following should be laid down for each quality:

- **scale** – the unit of measurement;
- **test** – the practical test of the extent to which the attribute quality exists;
- **worst** – the worst acceptable value;
- **plan** – the value that it is planned to achieve;
- **best** – the best value that appears to be feasible (the 'state of the art' limit); this would be a level that is known to have been achieved elsewhere;
- **now** – the value that applies currently.

In order to derive these *quality specifications* it is often necessary to break down a quality criterion into further sub-criteria. Take the quality criterion 'communicativeness', which contributes to the quality factor 'usability'. One aspect of this might be the ease of understanding of the menu structure, in particular, how easy it is to find the command to carry out some function. Another aspect of communicativeness would be how informative the error messages were, while yet another would be the clarity of the 'help' pages.

Suggest quality specifications for a word processing package. Give particular attention to the way that practical tests of these attributes could be conducted.

Exercise 12.4

12.5 ISO 9126

Over the years, various lists of software quality characteristics have been put forward, such as those of McCall, described above, and of Boehm. A difficulty has been the lack of agreed definitions of the qualities of good software. The term 'maintainability' has been used, for example, to refer to the ease with which an error can be located and corrected in a piece of software, and also in a wider sense to include the ease of making any changes. For some, 'robustness' has meant the software's tolerance of incorrect input, while for others it has meant the ability to change program code without introducing unexpected errors. ISO 9126 standard was published in 1991 to tackle the question of the definition of software quality. This 13 page document was designed as a foundation upon which further, more detailed, standards could be built.

ISO 9126 identifies six software quality characteristics:

- **functionality**, which covers the functions that a software product provides to satisfy user needs;
- **reliability**, which relates to the capability of the software to maintain its level of performance;
- **usability**, which relates to the effort needed to use the software;
- **efficiency**, which relates to the physical resources used when the software is executed;
- **maintainability**, which relates to the effort needed to make changes to the software;
- **portability**, which relates to the ability of the software to be transferred to a different environment.

ISO 9126 suggests sub-characteristics for each of the primary characteristics. It is perhaps indicative of the difficulties of gaining widespread agreement that these sub-characteristics are outside the main standard and are given in the document for information only. They are useful as they clarify what is meant by the main characteristics.

<i>Characteristic</i>	<i>Sub-characteristics</i>
Functionality	Suitability Accuracy Interoperability Compliance Security

Compliance refers to the degree to which the software adheres to application-related standards or legal requirements. Typically, these could be auditing requirements.

Interoperability and *security* are good illustrations of the efforts of ISO 9126 to clarify terminology. Interoperability refers to the ability of the software to interact with other systems. The framers of ISO 9126 have chosen this word rather than *compatibility* because the latter causes confusion with the characteristic referred to by ISO 9126 as *replaceability* (see below).

<i>Characteristic</i>	<i>Sub-characteristics</i>
Reliability	Maturity Fault tolerance Recoverability

Maturity refers to the frequency of failure due to faults in a software product, the implication being that the more the software has been used, the more faults will have been uncovered and removed. It is also interesting to note that *recoverability* has been clearly distinguished from *security*, which describes the control of access to a system.

<i>Characteristic</i>	<i>Sub-characteristics</i>
Usability	Understandability Learnability Operability

Understandability is a clear quality to grasp, although the definition 'attributes that bear on the users' efforts for recognizing the logical concept and its applicability' in our view actually makes it less clear!

Note how *learnability* has been distinguished from *operability*. A software tool might be easy to learn but time-consuming to use because, say, it uses a large number of nested menus. This is fine for a package that is used only intermittently, but not where the system is used for several hours each day by the end user. In this case, learnability has been incorporated at the expense of operability.

<i>Characteristic</i>	<i>Sub-characteristics</i>
Efficiency	Time behaviour Resource behaviour
Maintainability	Analysability Changeability Stability Testability

Analysability is the quality that McCall called *diagnosability*, the ease with which the cause of a failure can be determined. *Changeability* is the quality that

others have called *flexibility*: the latter name is perhaps a better one as changeability has a slightly different connotation in plain English – it implies that the suppliers of the software are always changing it!

Stability, on the other hand, does not mean that the software never changes: it means that there is a low risk of a modification to the software having unexpected effects.

<i>Characteristic</i>	<i>Sub-characteristic</i>
Portability	Adaptability
	Installability
	Conformance
	Replaceability

Conformance, as distinguished from *compliance*, relates to those standards that have a bearing on portability. The use of a standard programming language common to many software/hardware environments is an example of conformance. *Replaceability* refers to the factors that give ‘upwards compatibility’ between old software components and the new ones. ‘downwards compatibility’ is specifically excluded from the definition.

ISO 9126 provides some guidelines for the use of the quality characteristics. The fact that the relative importance of different quality characteristics will depend on the type of product under examination is stressed. Thus reliability will be of particular concern with safety-critical systems while efficiency will be crucial for some real-time systems. For interactive end user systems, the key quality might be usability. Once the requirements for the software product have been established, the following steps are laid down:

Quality metrics selection Measurements that correlate to the characteristics of each quality have to be identified. No specific guidance is given by the ISO 9126 standard on the applicability of the various measurements that might be used.

Ratings level definition The metrics used must be mapped onto scales that indicate the degree to which the requirements have been satisfied. For example, in one application ‘time behaviour’ in the sense of response time might be important. For a key transaction, actual response times might be mapped onto quality scores:

<i>response time (seconds)</i>	<i>quality score</i>
< 2	5
2-3	4
4-5	3
6-7	2
8-9	1
>9	0

Upwards compatibility means the capability for files created by the old version of the software to be used by newer versions.

Assessment criteria definition The way that the quality scores are combined or summarized to give an overall view of the product has to be defined. The software product has now to be evaluated by measuring its qualities, converting them to quality scores or ratings, and summarising the ratings to obtain an overall judgement. ISO 9126 does not specify how this has to be done, only that some method must be devised.

One approach, for example, recognizes that some quality rating levels will be mandatory. If a product fails to reach any of the mandatory rating levels it must be rejected regardless of how good it might be in other ways. Other characteristics can be desirable but not essential. For these desirable characteristics it is possible to give a rating in the range 1–5, say, reflecting how important they are. Above we have shown in the case of time behaviour how its quality in a particular product can be awarded a quality score on a scale 0 to 5. The scores for the more or less important qualities can be given due weight by multiplying each one by its importance weighting. These weighted scores can then be summed to obtain an overall score for the product. The scores for various products can then be compared to get a first cut order of preference. For example, the quality of two products might need to be compared on the grounds of usability, efficiency and maintainability. The importance of each of these qualities might be rated as 3, 4 and 2 respectively, out of a possible maximum of 5. Quality tests could result in the scores shown in Table 12.2.

Table 12.2 *Quality rating scores*

<i>product quality</i>	<i>importance rating (a)</i>	<i>product A</i>		<i>product B</i>	
		<i>quality score (b)</i>	<i>weighted score (a × b)</i>	<i>quality score (c)</i>	<i>weighted score (a × c)</i>
usability	3	1	3	3	9
efficiency	4	2	8	2	8
maintainability	2	3	6	1	2
overall			17		19

An attempt was made to use the ISO 9126 standard to define the qualities needed in the software to control systems in an unmanned station in the Antarctic that was to perform scientific experiments that were to be monitored and controlled by satellite link from Italy. Useful advice that came out of this was that the qualities required can vary from component to component within the system and that therefore quality definition at the level of the overall system is not always appropriate. The researchers also found that the most important quality for this application was availability, which was a mixture of the ISO 9126 top level qualities of reliability and maintainability – separating the two elements was in practice very difficult.

12.6 Practical software quality measures

Below are some ways of measuring particular qualities. It is emphasized that the measures are illustrations only and should certainly not be treated as definitive! Each project will need to have its own measures devised to meet its own specific needs. The measures described relate to the final software products of a project.

Reliability

This might be measured in terms of:

- **availability**, the percentage of a particular time interval that a system is usable;
- **mean time between failures**, the total service time divided by the number of failures;
- **failure on demand**, the probability that a system will not be available at the time required or the probability that a transaction will fail;
- **support activity**, the number of fault reports that are dealt with.

IOE maintenance group accounts system has been installed, and is normally available to users from 8.00 am until 6.00 pm from Monday to Friday. Over a four-week period, the system was unavailable for one whole day because of problems with a disk drive and was not available on two other days until 10.00 in the morning because of problems with overnight batch processing runs.

What were the availability and the mean time between failures of the service?

Exercise 12.5

Maintainability

This is closely related to flexibility, the ease with which the software can be modified. The main difference is that before an amendment can be made, the fault has to be diagnosed. Maintainability can therefore be seen as flexibility plus a new quality, diagnosability, which might be defined as the average amount of time needed to diagnose a fault.

Maintainability can be seen from two different perspectives. The user will be concerned with the *elapsed time* between a fault's being detected and its being corrected, while the programming management will be concerned about the *effort* involved.

Extendibility

This is a component of the more general quality of flexibility. It can be defined as the productivity needed to incorporate a new feature into an existing system expressed as a percentage of the normal productivity when developing the software from scratch.

The original IOE maintenance billing system comprised 5000 SLOC and took 400 work-days to implement. An amendment to the core system caused by the

Case Study Example

introduction of group accounts has lead to 100 SLOC being added which took 20 work-days to implement, thus:

$$\begin{aligned}
 \text{productivity for the original system} &= 5000/400 \\
 &= 12.5 \text{ SLOC/staff day} \\
 \\
 \text{productivity for the amendment} &= 100/20 \\
 &= 5 \text{ SLOC/staff day} \\
 \\
 \text{extendibility} &= 5/12.5 \times 100 \\
 &= 40\%
 \end{aligned}$$

12.7 Product versus process quality management

The measurements described above can be taken only after the system is operational. It might then be too late to do anything to remedy problems. What would be more helpful to someone like Amanda at IOE would be measurements and other checks that can be taken during development and that can help control what the final system will be like.

The system development process is made up of a number of activities that are linked together so that the output from one activity is the input to the next (Figure 12.2). Thus, program testing will depend on there being a program to test that will be the product of the program coding stage. Errors can enter the process at any stage. They can be introduced either because of a defect in the way a process is carried out, as when programmers make mistakes in the logic of their programs, or because information has not been passed clearly and unambiguously between stages.

Errors that creep in at the early stages are more expensive to correct at later stages, for the following reasons.

- The later the error is found the more rework at more stages of development will be needed. If an error in the specification is found at the testing stage, then this will mean rework at all the stages between specification and testing.
- The general tendency is for each successive stage of development to be more detailed and less able to absorb change.

Errors should therefore be eradicated by careful examination of the products of each stage before they are passed on to the next. To do this, the following *process requirements* should be specified for each activity.

- **Entry requirements**, which have to be in place before an activity can start. An example would be that a comprehensive set of test data and expected results be prepared and approved before program testing can commence.

This model should already be very familiar from the discussion of precedence networks where the dependence of an activity on the completion of one or more preceding activities is taken into account.

These requirements should be laid out in installation standards, or a *Software Quality Plan* can be drawn up for the specific project if it is a major one.

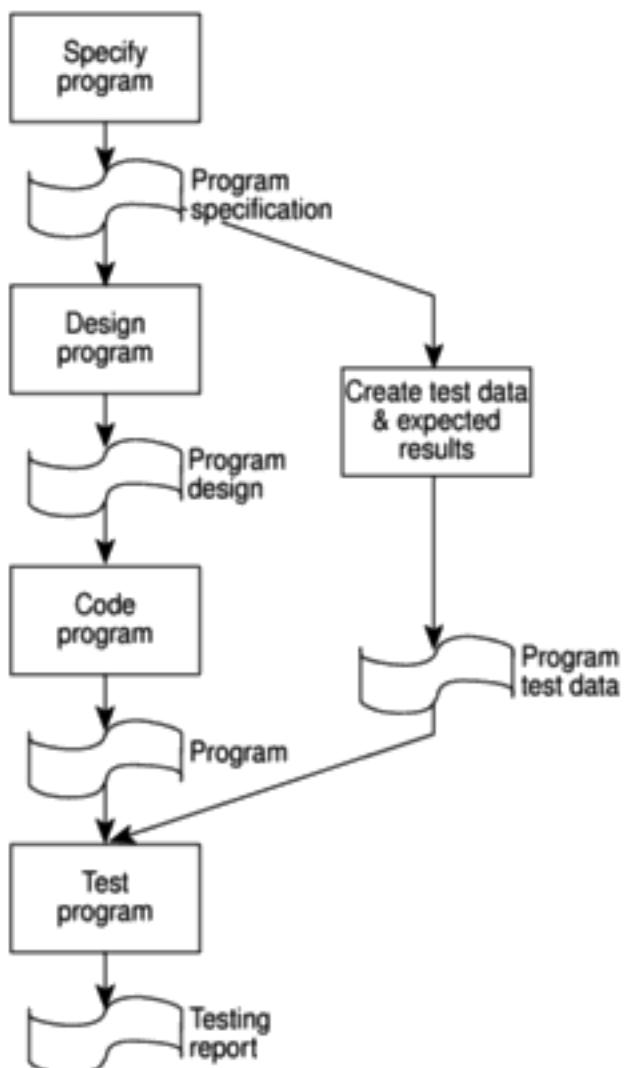


Figure 12.2 An example of the sequence of processes and deliverables.

- **Implementation requirements**, which define how the process is to be conducted. In the testing phase, for example, it might be laid down that whenever an error is found and corrected, all test runs must be repeated, even those that have previously been found to run correctly.
- **Exit requirements**, which have to be fulfilled before an activity is deemed to have been completed. For example, for the testing phase to be recognized as being completed, all tests will have to have been run successfully with no outstanding errors.

In what cases might the entry conditions for one activity be different from the exit conditions for another activity that immediately precedes it?

Exercise 12.6

Exercise 12.7

Amanda at IOE already has a quality manual that she can consult. Brigette at Brightmouth College has to specify her own entry and exit requirements. What might she specify as the entry and exit requirements for the process *code program* shown in Figure 12.2?

12.8 External standards

BS EN ISO 9001

The British standard for Quality Management systems was previously called BS 5750.

At IOE, a decision might have been made to use an outside contractor to produce the maintenance group accounts subsystem rather than develop the software in-house. As a client using the services of an outside contractor, IOE would be concerned that the contractor is following the best quality practices. It is now common to include in contracts terms covering the types of technique that a contractor will use. Various national and international standards bodies, including the British Standards Institution (BSI) in the United Kingdom, have inevitably become involved in the creation of standards for quality management systems. The British standard is now called BS EN ISO 9001:1994, which is identical to the international standard, ISO 9001:1994. Standards such as the ISO 9000 series aim to ensure that a monitoring and control system to check quality is in place. They are concerned with the certification of the development process, not of the end product, as in the case of crash helmets and electrical appliances with their familiar CE labels. The ISO 9000 series govern quality systems in general terms and not just those in the software development environment.

There has been some controversy over the value of these standards. Stephen Halliday, writing in *The Observer*, had misgivings that these standards are taken by many customers to imply that the final product is of a certified standard although as Halliday says 'It has nothing to do with the quality of the product going out of the gate. You set down your own specifications and just have to maintain them, however low they may be'. It has also been suggested that obtaining certification can be an expensive and time-consuming process that can put smaller, but still well-run, businesses at a disadvantage. Finally, there has been a concern that a preoccupation with certification might distract attention from the real problems of producing quality products.

Putting aside these reservations, let us examine how the standard works. A primary task is to identify those things that are to be the subject of quality requirements. Having defined the requirements, a system must be put in place to check that the requirements are being fulfilled and that corrective action is being taken where necessary.

An overview of BS EN ISO 9001 QMS requirements

Remember that these standards were originally designed for all kinds of production – not just software development.

In order for a quality management system (QMS) to meet the standard it has to conform to certain requirements which are summarized below.

- (a) The management must define and document the policy concerning quality and must ensure that this policy is communicated to all levels of the organization.
- (b) All quality control procedures must be documented.
- (c) All contracts to supply goods or services must contain mutually agreed requirements that the developer is capable of delivering.
- (d) There must be procedures to control and verify the design of the system to be supplied so that it meets the requirements agreed with the customer.
- (e) There must be procedures to approve design and other documentation.
- (f) Where components of the system to be supplied to the client are obtained from third parties there must be procedures to ensure, check and maintain the quality of these components.
- (g) Individual products must be identifiable as should their components.
- (h) The process by which the final product is created must be planned and monitored.
- (i) Inspection and testing must take place during the development phase, at its completion and before delivery. Tests and inspections must also be carried out on components obtained from third parties.
- (j) The equipment used in the production process itself must be properly controlled with respect to quality.
- (k) The testing status of all components and systems must be clearly recorded at all times.
- (l) Care must be taken to ensure that items that are known to be defective are not inadvertently used.
- (m) When a defect is detected, measures must be undertaken to remove the defective part and to ensure that the defect does not occur again.
- (n) Satisfactory procedures must be in place to deal with correct handling, storage, packaging and delivery of the product.
- (o) Sufficient records must be maintained to demonstrate that the quality system is working satisfactorily.
- (p) The software quality management system must be audited on a regular basis.
- (q) Servicing and support activities must be subject to the quality management system.
- (r) The developer must establish appropriate statistical techniques to verify the acceptability of the final product.

Exercise 12.8

Identify specific instances in a software development environment where the requirements about the control of equipment (j), the recording of the testing status of all components (k), and the correct handling, storage, packaging and delivery of the product (m) would be relevant. What procedures would apply in a software environment in relation to these requirements?

Exercise 12.9

Bearing in mind the criticisms of BS EN ISO 9001 that have been mentioned, what precautionary steps could a project manager take where some work of which the quality is important is to be contracted out?

TickIT

However, some parts do refer to software, for example, ISO 9000-3.

The ISO 9000 standards refer to quality management systems in general but in the United Kingdom, the Department of Trade and Industry (DTI) have formulated the TickIT standards which give an interpretation of these standards, which applies specifically to software development. This includes such requirements as:

- a detailed development plan is required before development is embarked upon;
- change control procedures should be used at all stages of development;
- design reviews must take place;
- the suitability of the design methodology must be reviewed;
- progress must be reviewed on a systematic basis;
- it must be possible to trace back the features of software design to specifications and requirements;
- designs must be properly documented;
- suitable test plans, specifications and records must be produced;
- a code of practice must be in place which governs the way the software is developed.

The code of practice must include the requirements that:

- the design must be broken down into levels, each with identifiable inputs and outputs;
- software must be organized into modules;
- a module must normally perform a single function or a set of related functions;
- a plain language description must exist for each module.

A TickIT auditor can certify that a particular organization conforms to these standards. This is called *certification*. The bodies doing this certification have to be *accredited* by the National Council for Certification Bodies (NACCB) on behalf of the DTI. The scheme is now administered by DISC, a part of the British Standards Institution.

Capability process models

Rather than just checking that a system is in place to detect faults, a customer might wish to check that a supplier is using software development methods and tools that are likely to produce good quality software. Even the TickIT recommendations can be regarded as fairly minimal. A customer will feel more confident, for instance, if they know that the software supplier is using structured methods. In the United States, an influential *capability maturity model* (CMM) has been developed at the Software Engineering Institute (SEI), a part of the Carnegie-Mellon University. This attempts to place organizations producing software at one of five levels of process maturity to indicate the sophistication and quality of their software production practices. These levels are defined as follows.

- **Level 1: Initial** The procedures followed tend to be haphazard. Some projects will be successful, but this tends to be because of the skills of particular individuals including project managers. There is no level 0 and so any organization would be at this level by default.
- **Level 2: Repeatable** Organizations at this level will have basic project management procedures in place. However, the way an individual task is carried out will depend largely on the person doing it.
- **Level 3: Defined** The organization has defined the way in which each task in the software development life cycle is to be done.
- **Level 4: Managed** The products and processes involved in software development are subject to measurement and control.
- **Level 5: Optimizing** Improvement in procedures are designed and implemented using the data gathered from the measurement process.

See Watts Humphrey,
Managing the Software Process, Addison-Wesley,
New York, 1989.

The SEI originally developed CMM for the US Department of Defense who wanted to be able to assess the capability of contractors from whom they procured software.

For each of the levels, apart from the default level 1, *key process areas* (KPAs) have been identified as distinguishing the current level from the lower ones. These are listed in the Table 12.2.

The assessment is done by a team of assessors coming into the organization and interviewing key staff about their practices using a standard questionnaire to capture the information. A key objective is not just to assess, but to recommend specific actions to bring the organization up to a higher level.

A criticism has been made of the approach that it is unrealistic to try and assess an organization as a whole – in reality there will be major differences between the way that individual projects are conducted. *Bootstrap*, which is a European initiative along the same lines as CMM, does allow assessment to be done at a project level.

Bootstrap also caters for ratings between the major levels, for example, at 2.6 which indicates that the project is better than level 2 but not yet up to a level 3 standard.

Table 12.3 CMM Key process areas

<i>Level</i>	<i>Key process areas</i>
1. Initial	not applicable
2. Managed	configuration management, quality assurance, sub-contract management, project tracking and oversight, project planning
3. Defined	peer reviews, inter-group co-ordination, software product engineering, integrated software management, training programme, organization process definition and focus
4. Managed	quality management, process measurement and analysis
5. Optimizing	process change management, technology innovation, defect prevention

Assessing software products

The concern in this section has so far been with the assessment of organizations and the processes that they use to produce software, but many purchasers of software, including project managers contemplating the purchase of software tools are more directly worried about the quality of the software product itself. Compilers for some programming languages, for example, are subject to certification. Much progress, however, has still to be made in this area.

12.9 Techniques to help enhance software quality

So far in this chapter we have looked at the steps a customer might take to ensure the quality of software produced by an outside supplier. We now need to look at what techniques a project team might wish to employ to help them improve their own software development processes. Three main themes emerge.

- **Increasing visibility** A landmark in this movement towards making the software development process more visible was the advocacy by the American software guru, Gerald Weinberg, of ‘egoless programming’. Weinberg encouraged the simple practice of programmers looking at each other’s code.
- **Procedural structure** At first programmers were more or less left to get on with writing the programs, although there might be some general guidelines. Over the years there has been the growth of methodologies where every process in the software development cycle has carefully laid down steps.
- **Checking intermediate stages** It seems inherent in human nature to push forward quickly with the development of any engineered object until a ‘working’ model, however imperfect, has been produced that can then be ‘debugged’. One of the elements of the move towards quality practices has been to put emphasis on checking the correctness of work at its earlier, conceptual, stages.

Gerald Weinberg, *The Psychology of Computer Programming*, Van Nostrand Reinhold, 1971.

The creation of an early working model of a system might still be useful as the creation of prototypes shows.

We are now going to look at some specific techniques in more detail. The push towards more visibility has been dominated by the increasing use of walkthroughs, inspections and reviews. The movement towards more procedural structure inevitably leads to discussion of structured programming techniques and to its later manifestation in the ideas of 'clean-room' software development.

The interest in the dramatic improvements made by the Japanese in product quality has led to much discussion of the quality techniques they have adopted such as the use of quality circles and these will be looked at briefly. Some of these ideas are variations on the theme of inspection and clean-room development, but they are seen from a slightly different angle.

Inspections

The principle of inspection can be extended to any document that is produced at any stage of the development process. For instance, test data needs to be reviewed – its production is usually not a high profile task even though errors can get through to operational running because of its poor quality.

The underlying procedure with inspections is that when a piece of work is completed, copies of the work are distributed to co-workers who then spend some time going through the work, noting any defects. A meeting is then held where the work is discussed and a list of defects requiring rework is produced. The work to be examined could be, for instance, a program listing that is free of compilation errors.

Our own experience of using this technique has been that:

- it is a very effective way of removing superficial errors from a piece of work;
- it motivates the programmer to produce a better structured and self-explanatory program because they know that other people will be criticizing it;
- it helps spread good programming practices because the participants discuss the advantages and disadvantages of specific pieces of code;
- it can enhance team spirit.

The main problem is maintaining the commitment of participants to a thorough examination of the work they have been allocated after the novelty value of reviews has worn off a little.

The item will usually be reviewed by colleagues who are involved in the same area of work, so that a programmer's work, for example, will be reviewed by fellow programmers. However, to reduce the problems of incorrect communication between different stages there will be representatives from the stages that precede and follow the one that produced the work under review.

IBM have put the review process on a much more structured and formal basis, and have produced statistics to show its effectiveness. A Fagan inspection (named after the IBM employee who pioneered the technique) is a much more formalized procedure which is led, not by the author of the work, but by a specially trained 'moderator'.

See M. E. Fagan's article 'Design and code inspections to reduce errors in program development' in *IBM Systems Journal* 15(3).

The general principles behind the Fagan method

- Inspections are carried out on all major deliverables.

- All types of defect are noted – not just logic or function errors.
- Inspections can be carried out by colleagues at all levels except the very top.
- Inspections are carried out using a predefined set of steps.
- Inspection meetings do not last for more than two hours.
- The inspection is lead by a moderator who has had specific training in the technique.
- The other participants have defined roles. For example, one person will act as a recorder and note all defects found and another will act as reader and take the other participants through the document under inspection.
- Checklists are used to assist the fault-finding process.
- Material is inspected at an optimal rate of about 100 lines an hour.
- Statistics are maintained so that the effectiveness of the inspection process can be monitored.

Exercise 12.10

This exercise needs to be done in groups. Select for review a small program that has been written by one of your colleagues.

Choose a moderator, a reader and a recorder for the group. Spend about 20 minutes examining listings of the program individually and then come together to review the code jointly.

Structured programming and clean-room software development

In 1968 E. W. Dijkstra wrote a letter to a learned computing journal which was entitled 'Go To Statement Considered Harmful'. This unfortunately led to the common idea that structured programming was simply about not using GO TOs.

One of the people most closely associated with the origins of structured programming is Dijkstra. In the late 1960s, software was seen to be getting more complex while the capacity of the human mind to hold detail remained limited. It was also realized that it was impossible to test any substantial piece of software completely – there were just too many possible combinations of inputs. The most that testing could do was prove the presence of errors, not their absence. It was suggested by Dijkstra and others that, because of this, the only way that we could reassure ourselves about the correctness of software was by actually looking at the code.

The way to deal with complex systems, it was contended, was to break them down into components that were of a size for the human mind to comprehend. For a large system there would be a hierarchy of components and sub-components. For this decomposition to work properly, each component would have to be self-contained with only one entry and exit point.

The ideas of structured programming have been further developed into the ideas of clean-room software development by people such as Harlan Mills of IBM. With this type of development there are three separate teams:

- a specification team, which obtains the user requirements and also a usage profile estimating the volume of use for each feature in the system;
- a development team, which develops the code but does no machine testing of the program code produced;
- a certification team, which carries out testing.

Any system is produced in increments, each of which should be capable of actual operation by the end user. The development team does no debugging; instead, all software has to be verified by them using mathematical techniques. The argument is that software that is constructed by throwing up a crude program that then has test data thrown at it and a series of hit-and-miss amendments made to it until it works is bound to be unreliable.

The incremental approach was discussed in Chapter 4.

The certification team carry out the testing, which is continued until a statistical model shows that the failure intensity has been reduced to an acceptable level.

As noted in Chapter 4, the object-oriented (OO) approach has now largely superseded the structured programming approach. OO can be seen as a development from the structured approach to software development and shows many of its concerns such as those for reducing coupling (data flows between components) and encouraging cohesion (placing associated software processes in the same place).

Formal methods

In the section above on clean-room development, the use of mathematical verification techniques was mentioned. These techniques use unambiguous, mathematically-based, specification languages of which Z and VDM are examples. They are used to define *pre-conditions* and *post-conditions* for each procedure. Pre-conditions define the allowable states, before processing, of the various items of data that a procedure is to work upon. The post-conditions define the state of those data items after the procedure has been executed. Because the mathematical notation is precise, a specification expressed in this way should be unambiguous. It should also be possible to prove mathematically (in much the same way that at school you learnt to prove Pythagoras' theorem) that a particular algorithm will work on the data defined by the pre-conditions in such a way as to produce the post-conditions. It needs hardly be said that in many cases this will be more easily said than done. In fact structured programming can be seen as an attempt to analyse a program structure into small, self-contained, procedures that will be amenable to this formal verification approach.

Software quality circles

Much interest has been shown in Japanese software quality practices. The aim of the 'Japanese' approach is to examine and modify the activities in the development process in order to reduce the number of errors that they have in their end products. Testing and Fagan inspections can assist the removal of errors – but the same types of error generally occur again and again in successive products

created by a specific type of process. By uncovering the source of errors, this repetition can be eliminated. To do this needs the involvement of all the staff in identifying the causes of errors.

Staff are involved in the identification of sources of errors through the formation of *quality circles*. These can be set up in all departments of an organization including those producing software where they are known as *software quality circles* (SWQC).

A quality circle is a group of four to ten volunteers working in the same area who meet for, say, an hour a week to identify, analyse and solve their work-related problems. One of their number is the group leader and there is likely to be an outsider, a *facilitator*, who can advise on procedural matters. In order to make the quality circle work effectively training needs to be given.

Problem solving by quality circles

The steps that the circle go through in solving problems are:

- (a) identify a list of problems;
- (b) select one problem to solve;
- (c) clarify the problem;
- (d) identify and evaluate the causes;
- (e) identify and evaluate the solutions;
- (f) decide on a solution;
- (g) develop an implementation plan;
- (h) present the plan to management;
- (i) implement the plan;
- (j) monitor the plan;
- (k) consider wider applicability of solution;
- (l) restart from (b).

Exercise 12.11

What are the important differences between a quality circle and a review group?

People often feel inhibited from contributing ideas to a group and brain-storming can help to reduce this inhibition.

A number of specific techniques characterize quality circles, the most prominent of which is *brainstorming*. A subject or problem for which ideas are needed is nominated and the group then suggest as many ideas as possible. As ideas are suggested they are written down on a flip-chart. Other members of the group do not, at this stage, make any comments or criticisms of any suggestions made. At the end of the session the group go through the ideas listed and put similar ideas together and combine overlapping ideas. Typically this technique

would be used to generate the initial list of problems or a list of possible solutions to a particular problem.

Also associated with quality circles is the compilation of *most probable error lists*. For example, at IOE, Amanda finds that the maintenance group accounts project is being delayed because of errors in the requirements specifications. The project team could be assembled and spend some time producing a list of the most common types of error that occur in requirements specifications. This is then used to identify measures that can reduce the occurrence of each type of error. They might suggest, for instance, that test cases should be produced at the same time as the requirements specification and that these test cases should be dry run at an inspection. Another result might be a checklist for use when conducting inspections of requirement specifications.

This exercise has to be carried out as a group. Select a particular area of common experience where problems have arisen in the past. For example, if you are a group of students you could use the course or module you are undertaking, or a recent assignment that you have just completed. By means of a brainstorming session, identify all the problems that the participants have had. At the end of the brainstorming session, group together similar problems and combine overlapping ones.

Exercise 12.12

The problems needing crisis management are often technical in nature. Quality circles can encourage problem solving activity at lower levels of the organization and prevent crises occurring as their causes are tackled.

The effectiveness of quality circles

For quality circles to work there must be full support for them at all levels of management. First-line management can feel threatened by them as they might appear to undermine their authority. After all, problem solving is one of their main tasks. The proponents of quality circles see them as a way of giving management more time for planning and development. Any manager will have to devote time to 'fire-fighting', dealing with ad hoc crises, and this can detract from longer term activities that will be able to improve the effectiveness of the organization.

The GQM approach

Approaches to process improvement, such as the quality circles that have been just discussed, may be supported by quantitative measurement techniques. One of these is the GQM (Goal/Question/Metric) approach. This requires that the *goal* to be achieved is firstly identified. A goal might be to evaluate whether a new programming language allowed developers to be more productive. In order to achieve this goal, certain specific *questions* now need to be identified as needing answers. For example, in order to evaluate the productivity of a new programming language the following questions might need answering:

- how quickly were developers previously writing code?
- how quickly are developers writing code with the new programming language?

For further details of GQM see V. R. Basili and H-D. Rombach 'The TAME project. Towards improvement-oriented software environments' in *Transactions on Software Engineering* Volume 14(6) pp 758–773. Also GQM by van Solingen & Berghout, McGraw-Hill, 1999.

- how good was the quality the software previously?
- how good is the quality of the software produced using the new programming language?

A European initiative, *ami* ('application of metrics in industry') adopts a similar approach to GQM.

For each question, a number of *metrics* will need to be identified as needing collection in order to answer the question. For instance, to find out how quickly developers have been writing code, development effort and the size of development tasks in terms of the number of function points to be produced might be needed.

12.10 Conclusions

Important points to remember about software quality include the following.

- Quality by itself is a vague concept and practical quality requirements have to be carefully defined.
- There have to be practical ways of testing for the relative presence or absence of a quality.
- Most of the qualities that are apparent to the users of software can be tested for only when the system is completed.
- Ways of checking during development what the quality of the final system is likely to be are therefore needed.
- Some quality enhancing techniques concentrate on testing the products of the development process while others try to evaluate the quality of the development processes used.

12.11 Further exercises

1. McCall suggests that simplicity, modularity, instrumentation and self-descriptiveness are software quality criteria, that is, internal characteristics that promote the external quality of testability.
 - (a) Explain what is meant by each of the four criteria above.
 - (b) Describe possible measures for each of the criteria.
 - (c) Describe practical ways in which the measures could be assessed.
2. Discuss how meaningful the following measurements are.
 - (a) The number of error messages produced on the first compilation of a program.
 - (b) The average effort to implement changes requested by users to a system.
 - (c) The percentage of lines in program listings that are comments.

- (d) The number of pages in a requirements document.
3. How might you measure the effectiveness of a user manual for a software package? Consider both the measurements that might be applicable and the procedures by which the measurements might be taken.
 4. What might the entry, implementation and exit requirements be for the process *design program structure*?
 5. Identify a task that you do as part of your everyday work. For that task, identify entry, process and exit requirements.
 6. What BS EN ISO 9001 requirements have a bearing on the need for an effective configuration management system?

Chapter 13

Small projects

OBJECTIVES

When you have completed this chapter you will be able to:

- avoid some of the pitfalls that occur with students projects;
 - produce an initial planning document for a small, IT-related project.
-

13.1 Introduction

Many of the readers of this book are students who will have to plan their own projects that they will have to carry out as part of their course of studies. In some cases, these will be undertaken for an external client. In other cases, a piece of software, perhaps of an experimental nature, is to be produced where there is no identifiable client, apart from a project tutor. Although the projects that are carried out for 'real' clients are more convincing tests of the student, they are in many ways more risky than the purely academic ones.

One of the problems that students face when planning projects is applying techniques that they have learnt on software project planning courses and that were designed for much larger-scale projects than their own. In this chapter we present an outline of how students should set out their plans. It is based on a structure that we have recommended to our own students over the years. It contrasts with the material in Appendix A on PRINCE 2, which is designed to support the management of large projects. The overall Step Wise approach is still applicable to the planning of your project – but the techniques used at the different steps of the planning process will need to be carefully chosen as appropriate to the scaled-down application.

Note that here we are discussing a practical application of the risk identification and avoidance policies.

13.2 Some problems with student projects

There are some problems or risks that seem particularly to affect student projects.

Use of unfamiliar tools

Very often students will be using a new software tool (for example, an application builder in a Windows environment) that they have not used before. Clearly, time will need to be allocated in the project plan to learning the package. When trying to formulate plans, because of their ignorance of the software tool, students might have difficulties estimating how long tasks will take. There will also be risks that unexpected technical problems will halt or delay the project's progress. Students often have other things on their minds, such as examinations, in the period leading up to the start of the project, but the risks of technical problems will be reduced if they are able to try out the software tools at this point.

If you are really keen to learn about and use a new tool, you might consider framing the objectives of the project to include an evaluation of the tool. Technical difficulties then become transformed from being obstacles to being interesting data.

Uncertain design requirements

Many project assignments require students to demonstrate a careful analysis and design process and then to build at least part of the application. Until the analysis has been done, it can be difficult to plan exactly how design and software building is to be executed.

Two points are worth making here. The first is that the structure of the project is going to be dictated to a large extent by the amount of time available.

Say that you have ten weeks in which to carry out a project that involves analysing user requirements, designing a new system and building it. It is not a bad idea to start planning on the basis of doing the project in nine weeks in order to allow for slippage. Working backwards from the end, of those nine weeks, it might be decided that the last two should be reserved for testing and evaluation. This may seem an excessive amount of time to some students, but with most assessment schemes your project is likely to gain really good marks if you can demonstrate that it is of good quality and that you have evaluated it carefully. You might decide to allocate proportions of the remaining first seven weeks to analysis, design and system building so that you get the following skeleton schedule:

- **weeks 1–2:** analysis;
- **week 3:** design;
- **weeks 4–7:** system building;
- **weeks 8–9:** testing and evaluation;
- **week 10:** contingency.

The actual breakdown will vary, depending on the circumstances of your particular project. A very well structured problem area means that you can spend less time on analysis, while, on the other hand, if you know the software building tools that you are going to use, you might decide to reduce the time for analysis in favour of devoting more effort to a more sophisticated operator interface.

The second point, and one that we have already emphasized, is that you should be prepared to delay planning a particular phase of a project in detail until more information becomes available. When you have completed your analysis phase at the end of the second week in the above plan, you should be in a much better position to plan the design and system building phases in more detail – you will know, for example, what the main transactions are going to be in an information systems application. You should also be in a position to cut down the scope of the application to fit the time available to build it!

In Chapter 6 we discussed the practice of design to cost, that is, designing the system to fit the resources available to build it.

Incomplete systems

Sometimes students simply run out of time and so do not have a working system to demonstrate. In the case of student projects, it is a good idea to try to arrange things so that you have something, even if it is not much, to demonstrate from a relatively early stage of the project. In the skeleton plan described above, it would not be a bad idea to have certain features of the application up and running after the first week or so of system building. Having these ‘in the bag’ so to speak, you can go on and add new functions or features, secure in the knowledge that at any point you will have at least something to demonstrate. You might find that you can even break the work down into increments to give you something similar to the following:

- **weeks 1–2:** analysis;
- **week 3:** design increment 1;
- **week 4:** build/test increment 1;
- **week 5:** design increment 2;
- **week 6:** build/test increment 2;
- **week 7:** design increment 3;
- **week 8:** build/test increment 3;
- **week 9:** evaluate complete system;
- **week 10:** writing up/contingency.

In Chapter 4 we discussed the incremental approach which is relevant here.

If something went wrong so that increment 3 could not be completed on time, you would still have increments 1 and 2 to demonstrate and in your project evaluation report you could describe your proposed design for increment 3.

In the skeleton plan above, we have allocated week 10 for writing up. As a general rule, it is best to try and write up as much of your project as you can as you go along. It will not save you any time if you leave it and you will be able to write more clearly and fully about your analysis process, for example, while it is still clear in your mind. Pausing to do this writing should also help you to reflect on what you are doing and help you consider coolly what needs to be done next.

Lack of commitment from clients

In most cases, where a project is for an external client, the student will not be being paid. The advantage of this is that the outside organization might be attracted to the chance of having a free resource and take on a project for a student about whose capabilities they know little or nothing. The danger is that because the resource is free, there will be little commitment from the client to the project. If they had to pay for the work, they would think more carefully about whether they really needed the work before agreeing to the project and they would pay attention to getting value for money.

There is no such thing as a completely free project, even when the student's efforts are free. The student might need access to hardware and software facilities to be provided by the client and even where this is not the case, the client needs to be prepared to give up some spare time to discuss requirements and to evaluate intermediate results. It has to be said that sometimes clients can let students down as far as this is concerned. Our advice to students would be to try and get to know your client thoroughly before the project starts. Try, for instance, to meet all the people in the organization who will be affected by your work before you actually get going on the project. Show a copy of your initial project plan to the client and get them to comment on it. Arrange beforehand to have regular meetings with them to discuss progress.

13.3 Content of a project plan

We discuss below what should be in your project planning document. It is suggested that the plans for small projects being done for outside clients should follow this format. Specific course requirements (for example, learning objectives) are not covered.

1 Introduction

In the introduction to any document, it is always a good idea to explain briefly what the document is and why it has been produced. The introduction to the project plan should also include:

- identity of client – that is, the organization or department for whom the work is to be done;
- short description of project – not more than two or three brief lines;
- identity of the project authority – the person or persons within the client organization who will have authority over the project's direction. It is essential that such an authority be identified.

2 Background

This includes:

- relevant information about the client's business;

- descriptions of the existing software/hardware environment;
- circumstances or problems leading to the current project;
- work already carried out in the area of the project;
- stakeholders in the projects, in other words, all those who will be affected by the project or who have some other interest in it.

3 Objectives of the project

These might already have been defined in a terms of reference (TOR) document. If so, then it can be attached as an appendix to the plan.

The objectives must define what is to be achieved and the method of measuring the extent of that achievement. One problem with small projects conducted by students is that the project's success is evaluated in course terms soon after its completion, whereas the project's true value to the users will take much longer to emerge (if ever!). As this document is in part for the benefit of the client, the objectives from the client's viewpoint should have the emphasis here.

Students sometimes have problems distinguishing between their personal objectives and those of the project. The project objectives are those that are held in common between the client and the developer in relation to the project. For example, a university student could be producing a timetabling system for a local school. The creation of this timetabling system would be a project objective shared by the developer (the student) and the client (the school). The student will also have a personal objective of getting a good grade that is not shared by the school (although they will usually be sympathetic to it). From the viewpoint of the student's personal objectives the creation of the timetabling system is seen as a sub-objective, or goal, a step on the way to achieving the overarching objective of getting the good grade.

Objectives and goals were discussed in Chapter 1.

Where there are several objectives, an order of priority should be given to them if this is possible.

4 Constraints

It is convenient to merge this into the project objectives, above. Constraints include:

- externally imposed time scales;
- legal requirements;
- specific standards;
- limitations on the people who can be approached for information.

5 Methods/technology to be used

These might have already have been laid down for you, or it might be that part of the project will involve the selection of the most appropriate technologies and methods. In other cases, you need to specify the general approach you will take –

for instance that you are going to use a structured systems analysis and design method (like SSADM), or a soft system methodology (like SSM) approach. In a small project it is unlikely that you will have time for a full structured analysis/design approach and you might decide to use a subset of the techniques – this should be specified.

The methods selected will, of course, govern much of what will go into Section 6 below on project products and Section 7 on activities. When discussing prototypes, we emphasized that a prototype should always be a tool for learning or clarifying something (for instance, the best interface for the user). If you claim you are producing a prototype, then you must be prepared to define learning objectives for the prototype, a method of evaluation and an analysis of what has been learnt.

If you are developing software, then the choice of software tools should be stated in this section.

Some justification of the decisions made here should be given, as they can be crucial to the success of the project.

The training and other resource requirements (perhaps the need to purchase a particular package) that result from the decisions should be noted.

6 Project products

This is a list of all the products or deliverables that the project will produce, such as software modules, documentation, user guides and reports.

Intermediate products, such as design documents, should be included.

7 Activities

This is a list of the main activities that the project will involve. There must be activities to produce all the products listed in 6. Also, in general, each activity should result in some deliverable: avoid tasks like 'familiarization with departmental procedures' in favour of 'documentation of departmental procedures'. In identifying activities, new interim products will often be discovered.

For each activity, define:

- **pre-requisites**, what has to be done before this activity can start;
- **dependent activities**, activities that need this one to be completed first;
- **estimated time/effort**, this may be a range of values;
- **quality checks**, details of how you are going to verify and validate the product of the activity.

PERT or Gantt charts may be used but are often not needed.

8 Resources

This includes staff time, accommodation and hardware/software requirements.

The drawing up of a product flow diagram helps to bridge between the product list and the activity plan – see Chapter 2.

9 Risk analysis

Identify the main things that can be seen as possibly going wrong. Typically this might include:

- unavailability of resources (as with a delay in getting a software package);
- unavailability of key client personnel;
- technical problems (such as software bugs).

A priority can be given to each risk by allocating a *probability rating* (1–10) and a seriousness of *impact rating* (1–10). Multiplying the two together gives an overall score for priority purposes.

For the most serious risks (those with the highest scores), *preventive* measures to reduce or remove the risk should be specified. For example, in order to prevent problems with the unavailability of key client personnel, meetings must be arranged with them at the planning stage and holiday plans may be ascertained. In some cases *contingency* measures that can be undertaken once the risk has actually materialized are more appropriate.

13.4 Conclusions

In this chapter, we have tried to show how some of the broader issues of software planning that have been covered elsewhere in the book can be related to the kind of task a student might be asked to undertake as a project. On the other hand, a student undertaking a substantial industrial placement might be involved in a project team undertaking a large project and using a method such as PRINCE 2, described in Appendix A, to control the project.

Important points to remember include:

- take care when using techniques or tools that are completely new to you in a project;
- you will have to adjust the scope of your project to fit the time available;
- take steps to maintain the commitment of the client to your ‘free’ project;
- recognize the distinction between the project objectives and your personal objectives;
- avoid planning activities for which there is no tangible result – like ‘familiarization’.

Appendix A

PRINCE 2 – an overview

A.1 Introduction to PRINCE 2

Large organizations can have a number of software and other projects being executed at the same time. Some of these might use external suppliers of products and services. In such an environment it would be helpful if the procedures by which each project were run were standardized rather than having to be continually re-invented. However, each project will make different demands on management: some, for example, might be more technically challenging, or might affect particularly critical areas of the business or might involve larger numbers of different types of users. Because the adoption of a management method is not cost-free, the degree of control that will be cost-effective will vary from project to project. Hence any standard approach should incorporate mechanisms to tailor management procedures and structures to suit localized needs. In the UK, the government has sponsored, through the CCTA, a set of such procedures, called PRINCE, which has, after several years, been revised as PRINCE 2.

The precursor to PRINCE was a project management method called PROMPT, which suffered from the defect that it was not flexible enough to deal adequately with all types of project. This was followed by the first version of PRINCE, which was designed primarily for an IT development environment so that, for example, it was made to have a good fit with SSADM. It soon became apparent, however, that the method was applicable to projects outside the strictly IT domain and PRINCE 2 makes no specific references to IT development.

It is now possible to take examinations in PRINCE 2 and to thus be recognized as a PRINCE practitioner.

The CCTA is the Central Computer and Telecommunications Agency, which, among other things, recommends standards for UK government IT projects. It is the owner of PRINCE. PRINCE® is the CCTA's registered trademark.

PRINCE stands for 'PRojects IN Controlled Environments'.

A.2 The components of PRINCE 2

The method does not claim to cover all aspects of project management. It has the following components:

- organization;
- planning;
- controls;
- stages;

- management of risk;
- quality;
- configuration management;
- change control.

The following list provides a convenient structure that we will use to explain PRINCE 2:

- techniques;
- organization;
- documentation;
- procedures.

This appendix will not explore the supplementary techniques where the PRINCE 2 manual lays down some basic requirements. We do not describe these areas in detail since there is sufficient material elsewhere in this book. They include:

- risk management;
- quality management;
- configuration management;
- change control

Below, we will outline the general approach to planning that PRINCE 2 advocates. It will be noted that PRINCE 2, compared, for example, to SSADM, is rather light in its description of techniques. It is stronger in its rules for the project management structures that should be adopted. In our view, its main focus is on the project as an information system. Project management information is identified and the procedures by which the various elements of this information are created, processed and used are described at some length. This project information is mainly associated with the delivery of *products*, in a controlled environment, resulting in benefits to the business.

A.3 PRINCE planning technique

The Step Wise approach was outlined in Chapter 2, while risk was discussed in Chapter 7.

Figure A.1 shows the stages in the planning process that are suggested by PRINCE 2. The first of these – *Design a plan* – is essentially deciding what kind of information is to go into the plan, particularly at what level of detail the plan is to be drawn. The remaining steps are very similar to the ‘Step-Wise’ approach to planning that was outlined in Chapter 2, except that in the ‘Step Wise’ approach the risk analysis was carried out immediately after the estimation of effort for each activity. This was because in our view risk identification follows on naturally from

estimation: you work out how long you think it will take to do an activity and then you consider what factors could work to make that estimate incorrect. Also, the identification of risks can lead to new activities' being introduced to avoid the risks occurring and this is conveniently done before the schedule is put together by allocating resources. However, the Step Wise approach is not necessarily at odds with PRINCE 2 as PRINCE 2 does emphasize the iterative nature of risk analysis.

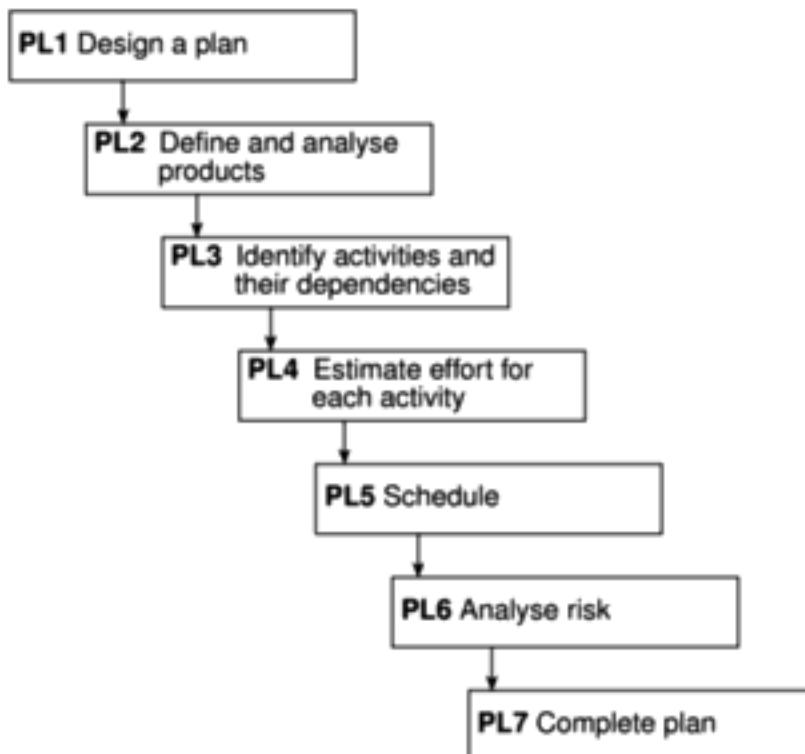


Figure A.1 PRINCE 2 approach to planning.

Like Step Wise, PRINCE 2 is very product-based. In the second planning phase, PL2 in Figure A.1, all the business products, plus the management and quality documents needed to control their delivery, are identified.

This sequence of steps can be used in several places in the PRINCE 2 procedural framework to produce a variety of different types of plan.

A.4 PRINCE 2 project organization

PRINCE identifies roles rather than jobs. Depending on the circumstances, a role could, in fact be carried out by more than one person, or a single person could assume more than one role.

PRINCE 2 is based on the perception that the project will involve *users* of the products of the project, on the one hand, and *suppliers* of goods and services needed by the project on the other. While the users and suppliers could in fact

belong to the same organization, for management and control purposes the two sides need to be carefully distinguished. Furthermore, on the customer side, two management roles exist. Any development project is carried out, not for its own sake but to do something useful for the customer organization. The *Executive* role has the responsibility of ensuring that the project continues to meet these business requirements. A danger, for example, is that development costs might grow in such a way that they exceed any benefits of the completed project. The customer side will also, of course, contain the community who will actually use the completed system on a day-to-day basis. Although we have talked about the supplier and customer sides, it could also be argued that the suppliers who will provide the system and the users who will operate it need to co-operate together to ensure that the operational system provides the benefits sought after by their customer, the 'Executive'.

PRINCE 2 specifies that the three roles of *Executive*, *Supplier* and *User* are represented on a *Project Board* which has overall accountability for the success of the project and responsibility for the commitment of resources.

The senior staff carrying out the respective roles will be responsible officers within their respective organizations and the oversight of the project will probably be only one of many responsibilities. Hence, the task of managing the project on a day-to-day basis will be delegated by the Project Board to a *Project Manager*. On a large project it could be necessary for the Project Manager to delegate the managing of certain aspects of the project to specialist *Team Managers*.

Conscientious and motivated staff will inevitably focus on meeting user requirements and give a lower priority to dealing with what they might see as project management 'red tape'. It could even be that the Project Manager with the daily burden of pushing the project forward might not be immune to this. However, this 'red tape' is needed to ensure that the project remains under control and that it continues to meet its business justification. Thus, some assurance is needed, independent of project management, that project management procedures are being properly followed. The ultimate responsibility for this assurance resides with the Project Board, but in practice detailed project assurance could be carried out by staff, independent of the team executing the project, who report to the Project Board members. Different types of project assurance specialists might be employed to ensure the business justification of the project is maintained, that the users' needs are being met and that the necessary technical requirements are being adhered to by the suppliers.

Note that we have followed the convention of indicating specific PRINCE 2 terms by initial capital letters, Project Board, for example. All these terms are as in the PRINCE 2 manual, which has Crown copyright.

The Project Board, Project Manager, Team Leaders and project assurance and support staff are known collectively in PRINCE 2 as the Project Management Team.

Project support

The Project Manager can require day-to-day support with the administration of the project. This might involve such tasks as processing time sheets or updating a computer-based project management tool such as Microsoft Project. It could be convenient for one group within an organization to supply this support to a number of projects. A key member of the project support team will be the Configuration Librarian, who will keep track of the latest versions of the products and documents generated by the project.

A.5 Project Stages

It is sensible to divide large projects into more manageable segments. PRINCE 2 caters for this through the idea of *Stages*. These are subsets of the project activities and are managed as sequences of individual units. Normally, the Project Manager will, at any one time, be authorized by the Project Board to execute only the current Stage. The Project Manager will be able to start the next Stage only when the Project Board has met to give its approval for the plans for that Stage. The end of a Stage signals a decision point when the Project Board will review the progress to date and reassure itself that the project is still viable from a business point of view – in particular, that the expected benefits are still likely to justify the projected costs.

The typical system development life cycle contains a number of phases, where each phase makes use of different specialist techniques. These technical phases might be the typical Waterfall steps outlined in Chapter 1: requirements analysis and specification, logical design, physical design, build, testing and installation. It is convenient in many cases for the management Stages specified by PRINCE 2 to be mapped onto these technical phases, but the PRINCE 2 standards are at pains to stress that it is not always convenient to do this – for instance the project might be more manageable if more than one technical phase were combined to create a Stage.

As will be explained in more detail in the following section on ‘Starting a project’, at the beginning of a project a *Project Plan* will be created which will give the envisaged Stages. Only the first of these Stages will need to have a detailed *Stage Plan* immediately available. For the later stages, it is better to complete the detailed Stage Plan just a little while before the Stage is due to start. In that way, the Stage Plan can take account of a more complete picture of the project: at the beginning of the project, for example, it would be impossible to plan the system building stage in detail when the system requirement has not yet been clearly defined.

Once the Stage has been authorized and its execution has been embarked upon, the Project Board should not need to meet as long as any deviations from planned time and cost are only minor and are within laid-down project tolerances. It should be sufficient for members of the Project Board to receive regular reports from the Project Manager. If the Project Manager becomes aware that these tolerances are likely to be exceeded, then they have a responsibility to produce an Exception Report for the Project Board. If the problems are serious enough to undermine the Stage Plan, then the Project Manager might then be required to produce a modified Stage Plan, or more properly an Exception Plan, which the Project Board will need to approve formally. In extreme circumstances the Project Board might at this point to decide to terminate the project prematurely.

A.6 Project procedures

Table A.1 lists the main project management processes for which PRINCE 2 lays

down procedures to deal with the various events that the Project Management Team might encounter.

Table A.1 Major PRINCE 2 processes

<i>PRINCE Id</i>	<i>Major processes</i>
SU	Starting up a project
IP	Initiating a project
DP	Directing a project
CS	Controlling a stage
MP	Managing product delivery
SB	Managing stage boundaries
CP	Closing a project
PL	Planning

The levels of staff who are involved with each of the groups of project management processes in Table A.1 are indicated in Figure A.2. The general planning process PL is not shown as this can take place at various times and places for different reasons. For example, it could take place during the ‘Initiating a project’ process to create the Project Plan, or during ‘Managing stage boundaries’ (SB) when a Stage Plan for the next Stage is constructed or when an Exception Plan needs to be produced.

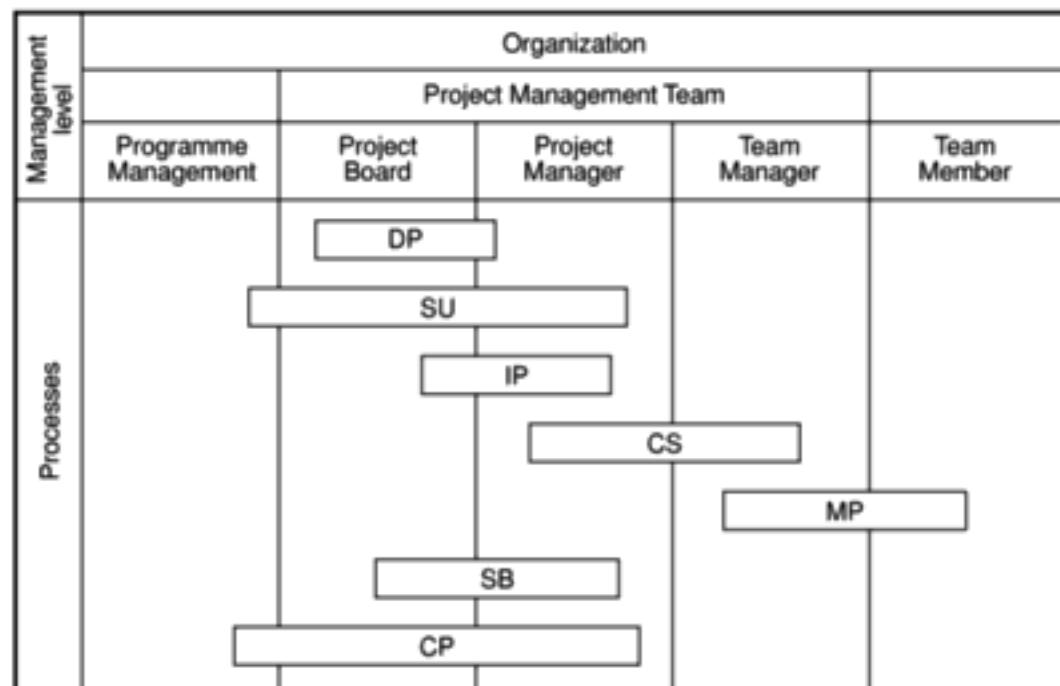


Figure A.2 Project management roles (see Table A.1 for key).

A.7 Directing a project

The main points where the Project Board have to be active are covered by the DP processes – *Directing a project*. These points are:

- authorizing initiation – agreeing to the start of detailed planning of a project;
- authorizing a project – agreeing, after the planning stage has been completed, that the project can go ahead;
- authorizing a Stage or Exception Plan;
- giving ad hoc direction;
- closing a project.

A.8 Starting up a project

As we noted in Chapter 3, the decision to undertake a project does not spring out of thin air. Where a customer organization has a coherent strategy, it is likely that there will be a layer of *programme management* where the ‘programme’ is a group of projects that are co-ordinated to meet an integrated set of business requirements. The current project could therefore be triggered by the programme managers. In any case, PRINCE 2 envisages that the project cycle will be sparked by some kind of *Project Mandate*, which will identify the customer and the general subject of the project. The PRINCE 2 Start Up process is essentially concerned with getting into a position where detailed planning of the proposed project can begin. As a starting point, this will need the recruitment of people to the various roles in the Project Management Team. The *Project Mandate*, which could be a rather insubstantial or imprecise document might need to be refined and expanded into a *Project Brief*, which defines the objectives of the project. Based on this, the general technical approach to be adopted to meet these objectives needs to be decided upon and documented in the *Project Approach*. The kinds of issue raised in Chapter 4 will need to be considered at this point. This could involve decisions about whether an off-the-shelf package can be bought or whether a bespoke package is required, and, if so, whether its development is to be carried out ‘in-house’ or by external contractors. All these activities lead to the formulation of a general plan of how the detailed planning is to be carried out.

With the first version of PRINCE there were sometimes difficulties knowing at which point the project had really started. The separate processes of Start Up and Initiation could avoid this.

A.9 Initiating the project

Having completed the Start Up Process, the Project Board can now decide that there are sufficient grounds to go on to more detailed planning. This begins with the consideration of a *Project Quality Plan*. Despite what some books suggest, quality levels do have cost implications. Different projects will have different quality requirements – faults in a college timetabling system are annoying but do not have the same consequences as the failure of a system controlling the flight of

a passenger aeroplane. These quality requirements will have an effect on the activities that will have to be scheduled and the resources that will have to be found. This Project Quality Plan, plus the information in the Project Brief and Project Approach documents, now allow a *Project Plan* to be drafted. This will contain:

- the major products to be created;
- the main activities to be undertaken;
- project risks and their counter-measures;
- effort requirements;
- timescales;
- key decision points.

We now have a much clearer idea of the overall cost of the project than we did at the time of the original Project Mandate. The business case can now be reviewed to see whether the proposed project is still viable. The reliability of the business case will depend on the validity of the assumptions upon which it is based. The possibility that particular assumptions are incorrect is assessed and documented in a *Risk Log*. The final parts of project initiation are specifying how the project is to be controlled in terms of reporting and decision-making responsibilities and the setting up of project files.

The culmination of the Project Initiation stage is the putting together of a *Project Initiation Document*, which brings together the documentation generated by the Start Up and Initiation processes. If the Project Board can approve this document then the first proper Stage of the project can start.

A.10 Controlling a stage

Once the Stage has been initiated, the Project Manager should be able to get on with the direction of the Stage without having to organize regular formal meetings with the Project Board.

Table A.2 shows the actions that the Project Manager might have to carry out while the Stage is being executed and for which the originators of PRINCE 2 have laid down procedural guidelines.

The Project Manager will have to authorize *Work Packages* (CS1), tasks that have to be carried out to create the products that should be the desired outcome of the project, such as software modules. On a substantial project, these authorizations will be passed not directly to the people who will do the work but to Team Managers.

Once the work has been authorized, the Project Manager will then need to find out how that work is progressing (CS2). This involves the kinds of task touched on in Chapter 9 on project control. For instance, progress information will have to be gathered to see if tasks are likely to be completed on time; feedback on recent

Table A.2 PRINCE 2 processes when controlling a Stage

<i>PRINCE ID</i>	<i>Controlling Stages (CS) processes</i>
CS1	Authorize work package
CS2	Assess progress
CS3	Capture project issues
CS4	Examine project issues
CS5	Review stage status
CS6	Report highlights
CS7	Take corrective action
CS8	Escalate project issue
CS9	Receive completed work package

quality checking activities will also be needed to ensure that apparent progress is not being made by releasing products before they are really ready. Eventually, for each work package the Project Manager will be informed that the work can be signed-off as completed (CS9). This progress data will be used to add actual completion dates to the details of planned activities that have been recorded in the Stage Plan.

A major part of a Project Manager's job during the execution of a Stage is bound to be 'fire-fighting' – dealing with the unexpected problems that are certain to occur. PRINCE 2 lays down a procedure (CS3 *Capturing project issues*) to ensure that these 'issues' are properly recorded. The 'issues' could be changes to requirements, changes to the environment such as new legal obligations, and other problems that might or might not have been foreseen by the risk analysis for the project. All these issues should be logged. Another PRINCE 2 procedure (CS4) is designed to ensure that all these issues are dealt with in an effective way. Outcomes can include a *Request for Change* to modify the user requirement or an *Off-Specification*, which records known and accepted errors and omissions in the product to be delivered.

The process of assessing progress (CS9) requires the Project Manager to look at the individual strands of work going on in his or her area of responsibility. PRINCE 2 envisages a separate but related activity where having gathered this progress information and also any outstanding Project Issues, the Project Manager checks the health of the Stage as a whole (CS5). In particular, the Project Manager will want to be reassured that the project as a whole is still progressing within its *tolerances*, the boundaries within which the Project Manager is allowed to manoeuvre without having to obtain clearance from the Project Board. One outcome of CS5 could be the carrying out of *Corrective action* (CS7) that might include authorizing new Work Packages to deal with specific problems. Where work is progressing so that Project Issues are being kept under control and the Stage is within tolerances, then it will be enough to communicate progress to the

Project Board by means of *Highlight Reports* (CS6). In some cases the Project Manager might feel unable to progress with a matter without guidance from higher management and will request advice. Where activities have taken longer than planned or have taken up more resources than were budgeted so that the Project Manager is in danger of having to act outside the tolerances laid down in the Stage Plan, the Project Manager might have to ‘escalate’ a particular issue (CS8) by drafting an *Exception Report* to be considered by the Project Board. This should not only explain why the Stage has gone adrift from its original plan, but also detail possible options for recovering the situation and make a specific recommendation to the Project Board.

A.11 Managing product delivery

The processes described in ‘Executing a Stage’ all assume that the work needed to complete a Stage is under the direct control of the Project Manager. Of course, it could be the case that, as described in Chapter 10 on contract management, some of the work is to be carried out by third party suppliers, that is, by an external organization that is not the primary supplier in direct contact with the customer, but a sub-contractor who carries out work on behalf of the supplier. These sub-contractors might not be using PRINCE 2. Hence the situation could need careful handling and PRINCE 2 provides some guidelines to help this – see Table A.3.

Table A.3 PRINCE 2 processes when managing product delivery

PRINCE Id	Managing product delivery (MP) processes
MP1	Accept work package
MP2	Execute work package
MP3	Deliver work package

Once the Project Manager has authorized a Work Package (CS1), as described in the ‘Controlling a Stage’ section, the person who is to be responsible for the execution of the Work Package needs to check the requirements of the Work Package to ensure that there is common understanding on what exactly is to be delivered, the constraints that might apply to the work and the requirements of any interfaces with other work (MP1). The Team Manager who is accepting the work must be confident that the targets can be realistically achieved. This could involve working out a *Team Plan* detailing how the work is to be done.

Once the work has been accepted, work can start on executing the Work Package (MP2). As this could be done by a sub-contractor who does not use PRINCE 2, PRINCE 2 lays down the general requirement that the responsible Team Manager should have the information ready to hand to report back to the Project Manager on progress as laid down in the authorized Work Package document. Finally, the need to define and agree the process by which completed Work Packages are handed over to the Project Manager is identified (MP3).

A.12 Managing stage boundaries

A key PRINCE 2 principle is to avoid too detailed planning at too early a stage. At the beginning of the project, for instance, the overall Project Plan is produced, but the more detailed Stage Plan is only produced for the first Stage. Towards the end of a Stage, the detailed plan for the next Stage can be mapped out as a clearer idea of the project requirements emerges (SB1). The creation of the Stage Plan for the next Stage could show up inadequacies in the overall Project Plan, which might need to be updated. For example, the design Stage of a project might reveal that the functionality of the system is greater than was foreseen when the first Project Plan was produced. More time might therefore be needed at the build Stage and this needs to be reflected in the Project Plan (SB2).

The transition from one Stage to another will involve the processes shown in Table A.4.

Table A.4 *PRINCE processes when managing stage boundaries*

PRINCE ID	Managing stage boundaries
SB1	Planning a stage
SB2	Updating the Project Plan
SB3	Updating the project business case
SB4	Updating the risk log
SB5	Reporting a Stage End
SB6	Producing an Exception Report

More time needed at build Stage will almost certainly mean that the date by which the project will be finally completed will be put back. This will lead to increasing development costs and the deferment of any income from the implemented system. At this point we need a process that checks that the project is still viable, that is, that the benefits of the delivered system will still outweigh the costs (SB3).

The situation with regard to risks might also have changed and this too needs to be reviewed (SB4). For example, as the project moves from design to build, some risks will disappear – if users were heavily involved in a design phase based around prototyping, a risk such as the non-availability of users for prototype evaluation will no longer be applicable and can be struck out. Other risks might, however, have materialized – a new version of the software building tool to be used could have been imposed by the organization and there is the possibility that developers might have technical difficulties adapting to the new product.

When all these things have been done, the new Stage will still need to wait for the successful completion of the last Stage. When this happens, the Project Manager can report the completion of the Stage (SB5) and the approval of the Project Board for the new Stage Plan can be requested.

A.13 Closing the project

PRINCE 2 divides the closing of a project into three separate processes:

- decommissioning a project (CP1);
- identifying follow-on actions (CP2);
- evaluating the project (CP3)

One follow-on action will be to plan for the Post Project Review which evaluates the effectiveness of the installed system after a set period of operation.

Decommissioning is mainly ensuring that all the loose ends are tied up. All Project Issues should either have been resolved or have been recorded as requiring *Follow-on Actions*. All the planned project products should have been accepted by the client and the requested operational and maintenance arrangements should be in place. Project files will have to be stored away into an archive and all parties involved should be notified that the project is now closed. PRINCE 2 does not specify that team members and key users should have a celebratory drink, but now might be the time to consider this. Decommissioning might have been caused by an ad hoc direction (DP4) to terminate a project prematurely as it is no longer required and in this case a wake could be more appropriate.

In organizations where development resources are scarce, there might appear to be little time available to reflect on practice and to dwell on past mistakes. However, if this is not done then time will be wasted in dealing with recurrent problems. PRINCE 2 recognizes this by specifying that at this point a *Project End Report* should be produced, documenting the extent to which the project has met the objectives set out in the Project Initiation Document, and also a *Lessons Learnt Report*, which should make suggestions about how problems could be avoided in future projects.

Appendix B

BS 6079:1996 – an overview

B.1 Introduction

Both as individuals and as organizations, we are continually exchanging goods and services – this is very much the basis of the commercial world we know today. This free exchange would be impeded unless on the whole there were a basic trust and common expectations about the nature and quality of the products involved and the processes by which those products had been created. To this end, business communities and governments have worked to produce standards. An important part of this has been the definition of agreed terminology so that we have a clear idea of what is meant by particular terms.

We would hope that standards are reasonably precise and unambiguous while being flexible enough to deal with the vagaries of the real world. It would also be hoped that standards would reflect accepted current best practice. A desirable feature of such standards would be some kind of accreditation of qualified practitioners so that customers for a product or service could know that they were dealing with someone competent in the particular area of expertise.

The British Standards Institution has had a leading role in the United Kingdom in this field and in 1996 published BS 6079 'Guide to project management'. While this 49-page document may have some claim to approach being a 'standard' because it reflects what the compilers of the document feel is current best practice, it is really, as its name suggests, a set of guidelines that are often couched in general terms of broad advice. In many ways it is like a general essay on project management.

The advantage of its being in the form of a British Standard, as Adrian Dooley has pointed out, is that this can give these broad, but important, project management principles more credibility in some quarters of the business community. A disadvantage of this format in our view, is that apart from other BSI standards, the project manager reading this text is given no information about where further guidance on the techniques described can be obtained.

A key question is how this standard fits in with the other UK standard PRINCE 2 or even why two separate standards are necessary. It is emphasized by the promoters of these two standards that they are not meant to be competitors. Pippa Newman, a CCTA Associate involved with promoting PRINCE 2 has written of PRINCE 2 that 'one of its major roles is to provide a means by which the British Standard can be implemented'. As will be seen, this could imply a

Adrian Dooley 'BS 6079 A base for the future?' in *Project Manager Today* 9(4) pp12-13, April 1997.

Pippa Newman 'PRINCE 2 The method for the next millennium' pp 14-15 of the same issue of *Project Management Today* as above.

degree of co-ordination between the developers of the two approaches that is not evident when the two documents are closely studied. One of the reasons for the differences in the two approaches might be their different origins – although PRINCE 2 is now meant for a rather wider context, its origins are in IT/IS development, while BS 6079 has more of a hard engineering background. However, once again, as Adrian Dooley commented:

‘With a bit of work an organisation could be consistent with both. The biggest problem is the need to harmonise the differing terminologies used by the two approaches. As someone who is keen to see project management develop as a mature discipline, I feel terribly frustrated by the lack of consistency in the profession. Why is it that the two bodies developing tools to promote better project management, both funded by the government, producing their respective documents at about the same time, cannot work closely together to agree consistent terminology at even the simplest level?’

Some of the differences between PRINCE 2 and BS 6079 are summarized in Table B.1.

The main elements of project management dealt with by BS 6079 are:

- the project life cycle;
- project organization;
- the planning process;
- project control;
- supporting techniques.

We will now look at each of these in turn.

The project life cycle

BS 6079 sees a ‘project’ in terms different from those of PRINCE 2. It uses the word to refer to the whole of the system life cycle from the initial idea right through the system’s operation to final decommissioning. The BS 6079 definition of project is similar to the view of a ‘project’ used in Chapter 3, when the cash projections for alternative ‘projects’ were calculated. When developing information systems, however, the development project is more usually seen as starting after the feasibility study has established that the project appears to be worthwhile and finishing when the system is handed over for operation. In fact, as Euromethod will demonstrate, it is possible to treat individual phases within that project as projects in their own right. As it happens, despite the BS 6079 definition of project, most of the BS 6079 guidelines for planning and control are compatible with the information systems development view of the project being completed at system hand-over.

Figure B.1 illustrates the main project phases as seen by BS 6079. Note that ‘implementation’ in this context means ‘implementation of the project plan’. Information systems developers often use the term ‘implementation’ to refer merely to the installation of the system once it has been developed. A further interesting point is that while the standard recommends the equivalent of the

Table B.1 Comparison of PRINCE 2 and BS 6079

<i>Topic</i>	<i>PRINCE 2</i>	<i>BS 6079</i>
Main focus	Procedures	Techniques
Definition of project	Temporary organization to deliver a business product	Covers the <i>whole</i> system life cycle
Project organization	Supplier vs Customer focus	Functional departments vs. project team
Project authority	Project Board	Sponsor
Stage organization	Detailed planning done by Stage	Concept of phases/milestones matches Stages – but no focus on incremental planning at phase level
Planning method	Product-driven (Product Breakdown Structures)	Task-driven (Work Breakdown Structures)
Work definitions	Product descriptions in the plan – turned into Work Packages by Project Manager	Statements of Work (SOW) in the plan
Techniques	Only minimal reference to techniques (such as to Activity Networks)	Detailed descriptions of techniques like Earned Value Analysis
Supporting techniques	Configuration, quality and risk management	those for PRINCE 2, plus financial control and procurement
Financial control	Cost control not dealt with in detail	Financial control dealt with more fully

PRINCE 2 Project End and Lessons Learnt Reports, no mention is made of the Post Implementation Review, or Post Project Review as it is now called.

Project organization

BS 6079 does not have PRINCE 2's emphasis on the Supplier–Customer relationship. Neither is the concept of a Project Board or steering committee presented. The focus of attention is on the project as a set of activities that cut across the normal functional structures of most organizations. A heavy emphasis is put on the desirability of a matrix management structure, where staff belong to a particular functional group in the longer term but may be allocated from time to time to multi-disciplinary teams that have been given the responsibility for achieving the objectives of a particular project.

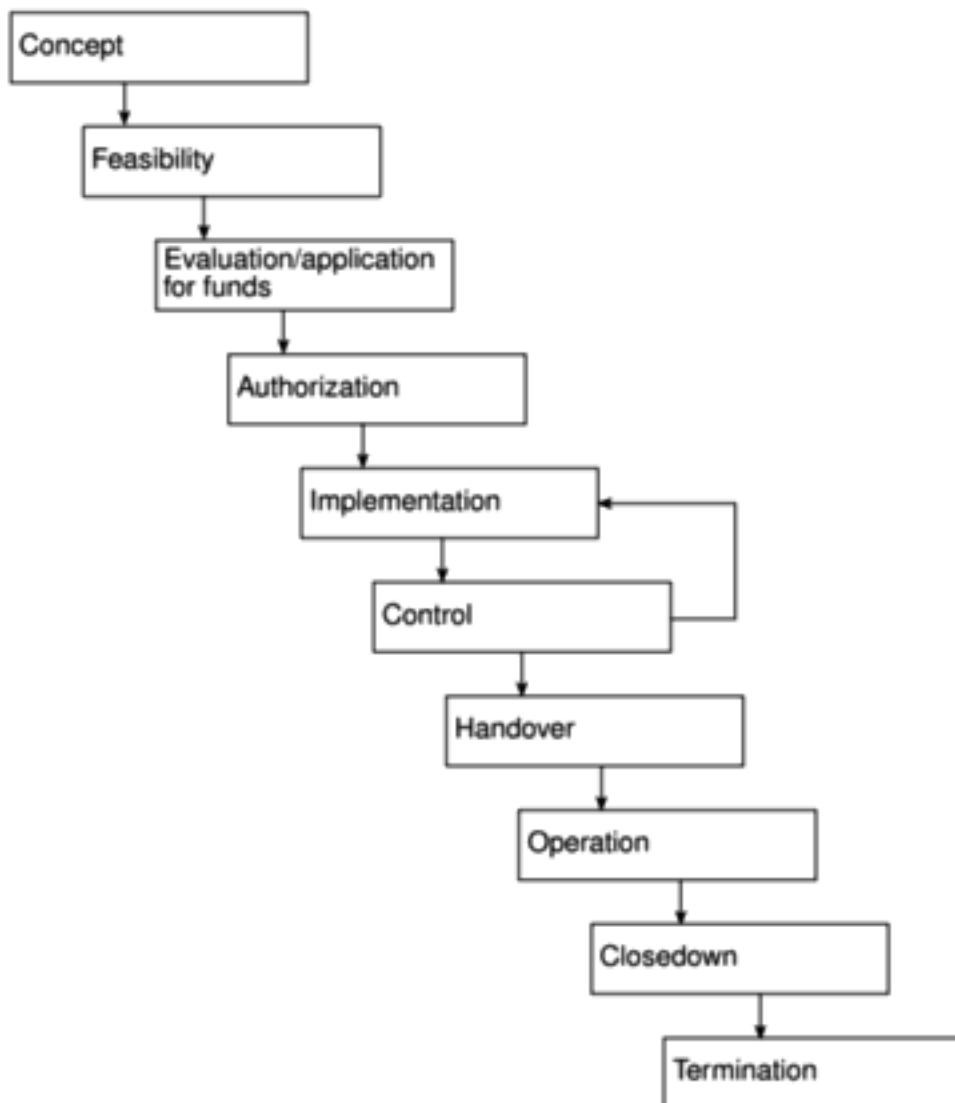


Figure B.1 BS 6079 project life cycle.

It can be argued that the need to create an environment for successful projects, which can take time, is an important message for the perceived readership of BS 6079.

One usually expects standards to be applicable fairly immediately, but the writers of BS 6079 warn that it might take an organization three to five years to move from a totally functional to a matrix organization. Rather surprisingly for a document that is supposed to be focused on project management, there is a section on the broader topic of how to bring about organizational change. This is an example of where a reference to a fuller treatment in another text would have been useful.

The other management role that BS 6079 emphasizes is that of *task owners*. Each task that is identified as needing doing in a project requires an owner, someone who will be accountable for its successful completion. Figure B.2 shows the relationships among the different project roles as envisaged in BS 6079.

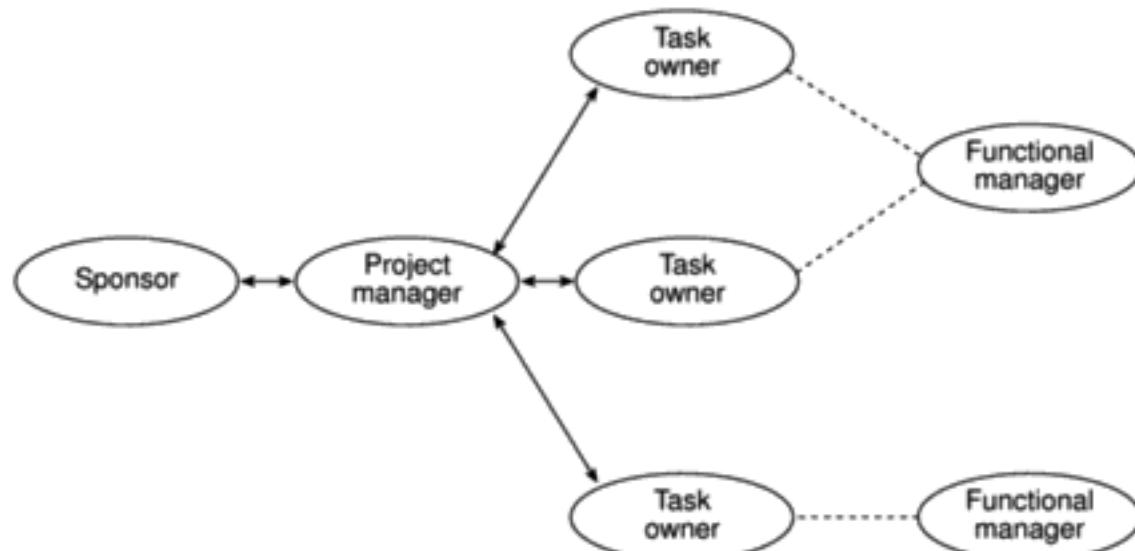


Figure B.2 BS 6079 project roles.

B.2 The planning process

According to BS 6079, a plan should have five main elements:

- introduction;
- commitment acceptance;
- work breakdown structure;
- schedule;
- statements of work.

The terms ‘work breakdown structure’ and ‘statement of work’ (SOW) will be explained later in this section.

One of the major differences between BS 6079 and PRINCE 2 is that while PRINCE 2 advocates a product-driven approach to planning, where the products the project has to create are initially identified, BS 6079 uses an activity-based approach, featuring a *work breakdown structure* (WBS), right from the beginning.

The standard does not put the same heavy emphasis on a Stage structure for projects as PRINCE 2. Passing references are, however, made to having project phases with milestones between the phases where there would be critical reviews of the progress that has been made and where decisions would be made about the continuing viability of the project. However, basing planning cycles on these phases, as PRINCE 2 does, is not suggested.

Apart from the WBS, the other important element in project documentation, according to this standard, is the *statement of work* (SOW), which combines elements of PRINCE 2’s Product Description and Work Package. A SOW is

BS 6079 does mention a ‘product-based WBS’ which equates to a PRINCE 2 ‘Product Breakdown Structure’, but does not use Product Flow Diagrams

created for each element in the WBS. These documents contain sections that cover such details as:

- summary of requirements;
- task ownership;
- key deliverables and their delivery dates;
- task dependencies;
- cost breakdown;
- risk assessment;
- performance criteria;
- progress reporting arrangements.

B.3 BS 6079 planning steps

The major BS 6079 planning steps are summarized below:

- **obtain project authorization;**
- **establish project organization.**

Develop WBS It is suggested that it is useful to produce a cross-reference table showing which activities are the responsibility of which parts of the organization.

Analyse project tasks This includes the identification of the task owner; of the way that the performance of the task is to be measured, its criticality to the project, its cost elements and any risks.

Assign task owner The importance of having people who have responsibility and accountability for the successful completion of each project task is very strongly stressed in BS 6079.

Develop the statements of work

Balance time, cost, integrity of specification and risk This is where the planner tries out different options for allocating effort and time-scale targets to activities to see what the likely consequences would be. Although Monte Carlo simulation is not mentioned by name, its possible use is implied.

Obtain commitments to do project tasks Task owners are asked to agree formally and commit themselves to the relevant tasks. Before the task owners can do this, they will need to quantify the resources that will be needed and to match this requirement against available resources, reserving the selected resources for use in the project.

Finalize agreements

Monte Carlo simulation was discussed briefly in Chapter 7.

B.4 Project control

Project control in BS 6079 is considered under the three distinct headings of change control, financial control and progress monitoring.

Change control BS 6079 recognizes that it is essential to have a formal mechanism to control changes, but while the requirements of such an administrative system are outlined, it does not specify any detailed steps. A major change can call for the planning cycle to be re-executed.

Financial control BS 6079 makes more specific references to financial control than PRINCE 2. The need is recognized for a process whereby the expenditure by the sponsoring organization on the project can be allocated to annual budgets. As time passes and progress is made, arrangements must be in place for the required funding to be released. Even with the best regulated projects, circumstances can require additional expenditure for unforeseen contingencies and a management reserve ought to be available for this.

Monitoring progress The BS 6079 emphasis on financial control is seen once again in the requirement that the project manager has a prime responsibility to report actual costs compared to planned costs. Task owners are required to pass on to the project manager revised estimates of the overall cost and projected completion dates for each task for which they have responsibility.

The compilers of BS 6079 clearly had a very strong preference for earned value analysis and the method is given considerable space in the standards document.

Not only does the project manager receive reports of progress from task owners, but he or she should warn the task owners of any external threats to their areas of activity. A report of a delay by one task owner can require the owners of tasks dependent upon the delayed task to be warned.

The need for risks to be continually assessed is stressed. Risk assessment techniques based on statistical probability are recommended.

The need for project managers to motivate and, if needs be, negotiate with, task owners is recognized by the document. Thus having good communications is stressed, and readers are warned that specifying additions to requirements is easier before a contract is signed than afterwards!

Earned value analysis has been described in Section 9.6.

The kind of risk management techniques that BS 6079 recommends have been discussed in Chapter 7.

B.5 Supporting techniques

It will be recalled that PRINCE 2 described the basic requirements for some supporting techniques, in particular, change and configuration management, quality management and risk management. BS 6079 adopts a similar approach but adds procurement and financial control.

With regard to procurement, the different organizational arrangements for making purchases for a project are outlined. For example, the buyers might belong to a centralized group or might be dedicated to the specific projects. The key role of purchase order documents as a control mechanism is highlighted. The need to

have some way of identifying trustworthy and qualified vendors, perhaps by maintaining a list of preferred vendors who have a satisfactory track record, is described in the BS 6079 document. The document also warns of the need to have in place procedures to expedite external work and to ensure the satisfactory standard of delivered work through inspections.

The section on financial control illustrates quite nicely the differences in approach between BS 6079 and PRINCE 2. In the BS 6079 document, two very specific quantitative techniques are described in some detail, namely the calculation of the net present value of a potential project and the analysis of the earned value of a project currently being executed. PRINCE 2 tends to avoid describing detailed technical data manipulation, with some notable exceptions to do with core planning methods, and concentrates on the necessary administrative procedures that need to be in place.

Appendix C

Euromethod – an overview

C.1 The aims of Euromethod

Euromethod (EM) has been around in draft form for some time as Version 0, but in July 1996 was officially released as Version 1. There were originally some misconceptions about what EM was, including that it was some kind of new analysis and design ‘supermethod’ that subsumed all the existing, well-loved, approaches such as SSADM, Information Engineering, Merise, SDM and so on. In fact, the problem that EM tries to tackle is that, within a unified European market, the existence of local standards and methods can effectively restrict the companies that can tender for contracts to develop systems to local organizations. Some action had already been taken so that a public body could not specify that a supplier had to use SSADM, for example, but rather had to specify ‘SSADM or equivalent’. However the question remained: what exactly was equivalent to SSADM? This is one of the questions that EM attempts to answer.

As the EM document states:

‘In the field of information systems, an open market can only be fully achieved by removing obstacles to the mutual understanding between customers and suppliers from different countries. These obstacles stem, in part, from the existence of a large variety of methods, each having its own concepts and vocabulary.’

EM works at a higher management level than conventional project management methodologies, but can have a decisive influence on the way that those methodologies are applied. EM’s concerns are primarily at the *contractual level*. As such, its focus is on the customer-supplier relationship and on *acquisition management*, see Figure C.1.

Euromethod attempts to assist mutual understanding between customer and supplier by providing guidance about how their relationship should be governed. A key feature of its approach is the way its guidance is based on an analysis of the problems and risks inherent in the current environment. It also attempts to provide a framework for the harmonization of terminology, although its effectiveness here is compromised by its adoption of a rather strange set of ‘Eurospeak’ terms.

The Version 0 EM documentation came to 800 pages. Version 1 is a mere 222 pages but is not always an easy read. It is however freely available over the Web.

EM has provision for the acquisition of both products and services. In this overview we will focus on products only.

C.2 The basic Euromethod model

An organization is perceived as having a number of *information systems*. These systems will have some procedures that are carried out by people and other

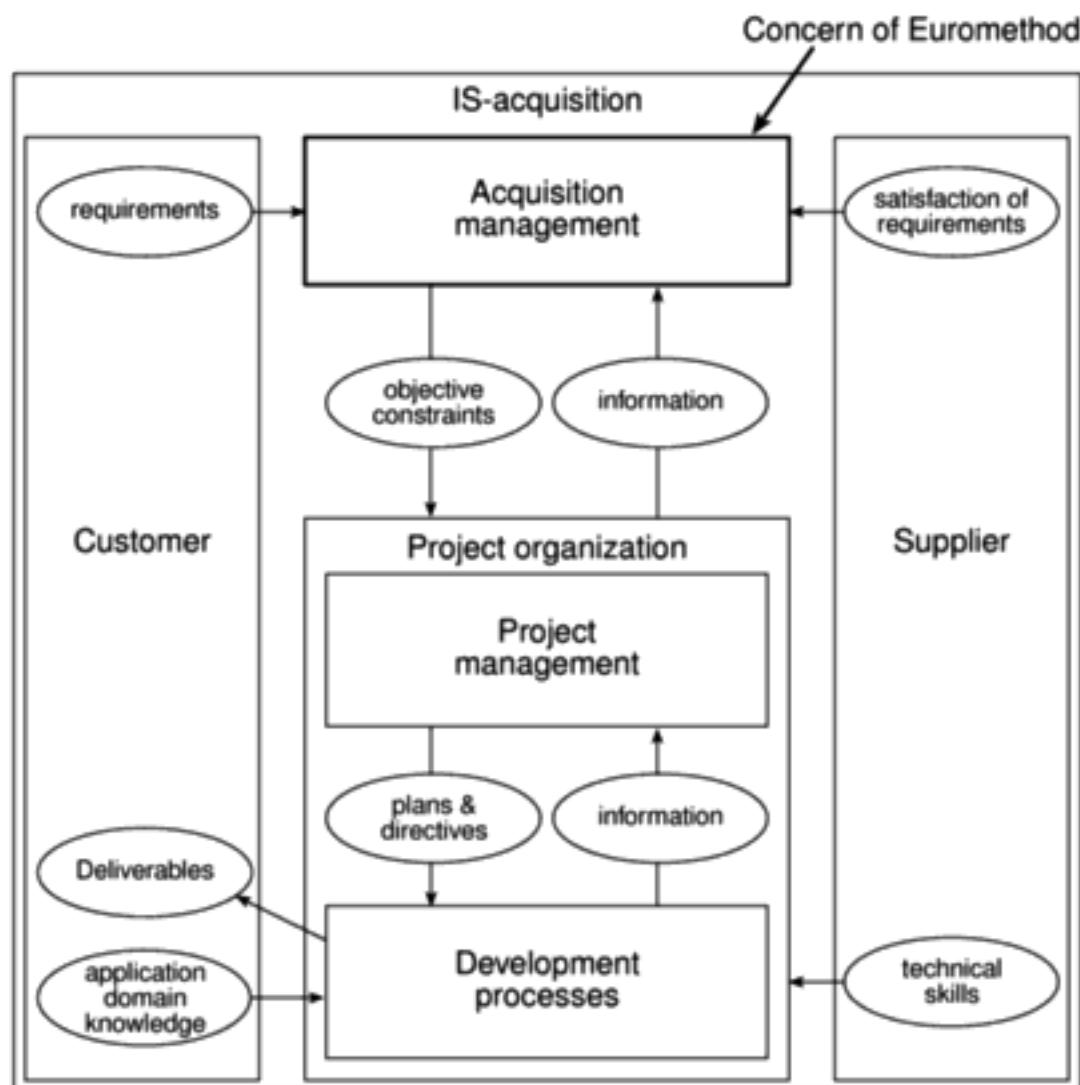


Figure C.1 Euromethod's area of concern.

elements that are executed via *computer systems*. An information system, possibly with its embedded computer system, might need to be changed from time to time and these changes are implemented by means of *adaptations*.

The basis of the idea of an *adaptation* (as illustrated in Figure C.2) is that an information system – and the knowledge that exists about that system – in a defined *initial state* is transformed, along with the knowledge about it, into a defined *final state*. For example, the initial state could include the current manual system and a requirements definition for a new, automated, system and the final state could be the new system after installation. With very large systems,

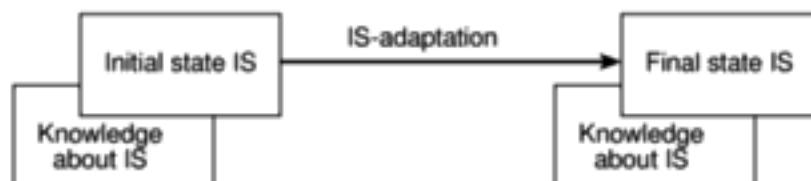


Figure C.2 Initial and final IS states.

development can be by means of a sequence of adaptations where the final state of one adaptation could be the initial state for the next – see Figure C.3. A danger is that a single adaptation might be too ambitious and EM provides guidance about where adaptations might be usefully split. One or more adaptations would be the subject of a contract that would have associated with it a *procurement process* where a supplier would be selected and a contract agreed, and the contracted-for products and/or services eventually delivered.

EM aims to produce a plan that punctuates these processes with a number of carefully defined *decision points* where the customer, in co-operation with the supplier, has to make decisions about the direction of the adaptation. These decision points will usually involve the supplier's delivering some product to the customer and this product forms the basis upon which the decisions can be based, such as system specifications that need to be approved.

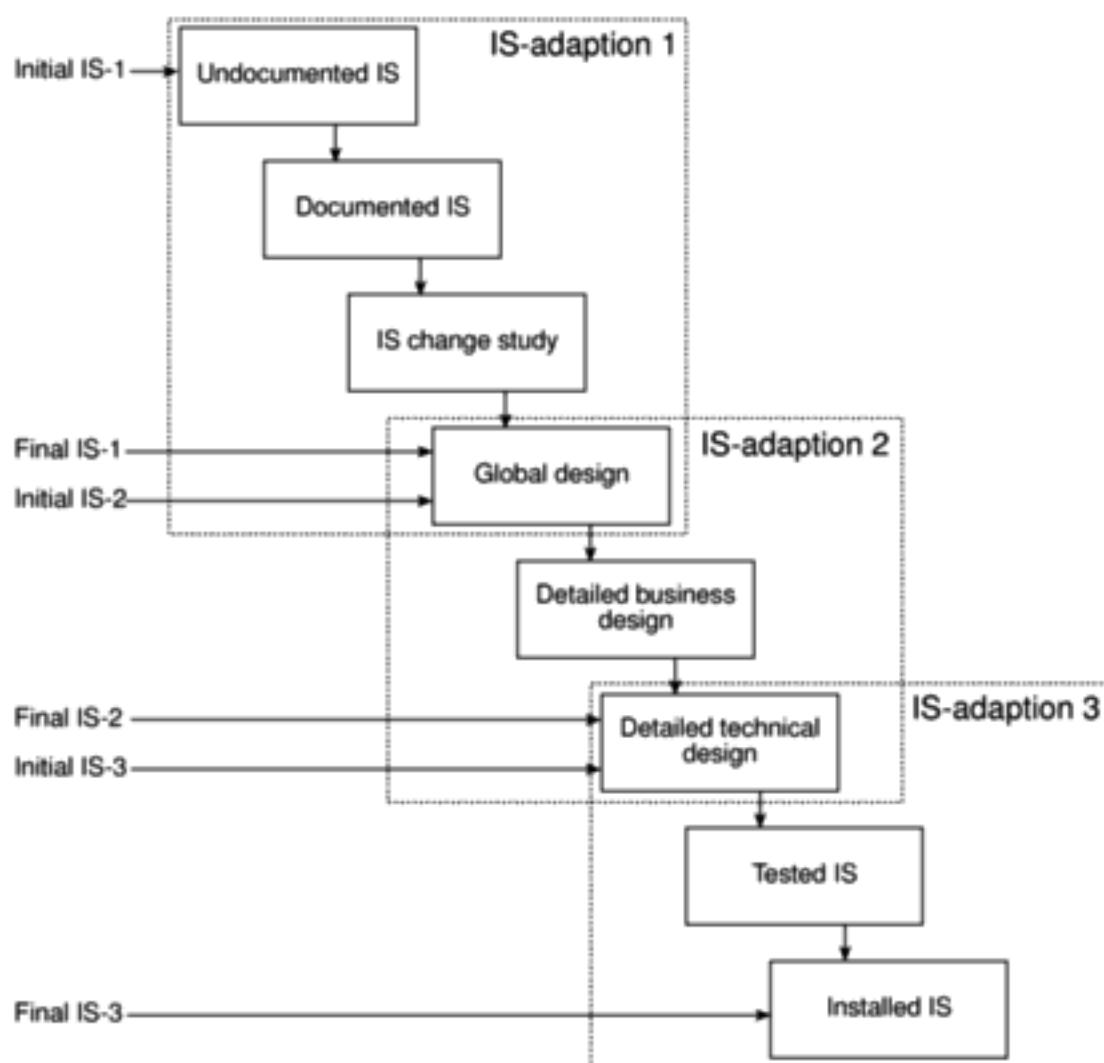


Figure C.3 A sequence of IS-adaptations.

C.3 An overview of the EM acquisition process

In PRINCE 2, a 'programme' is a portfolio of projects designed to achieve a set of defined business needs.

In this section, we are going to look briefly at the EM acquisition process. It is stressed that this is only an overview and anyone wanting to use the approach in earnest needs to look at the original EM documentation. In EM terms, 'acquisition' refers to the whole set of activities that a customer should go through to obtain a system or service. In some ways the EM acquisition process is analogous to a programme in a PRINCE 2 environment, in that it can be made up of a number of conventional projects. The system can have several components and some or all of these can be acquired by placing a contract with a supplier. *Procurement* refers to the preparation of an individual contract and the subsequent activities that ensure that the requirements of the contract are fulfilled. The contracted work can involve one or more 'adaptations' and for each of these, an *adaptation plan* will need to be devised and agreed by the supplier and customer. We will look at adaptation planning as a separate topic in its own right in a subsequent section.

The acquisition process starts with two steps: *acquisition goal definition*, which is broadly concerned with what is required; and *acquisition planning*, which is concerned with how the requirement is to be achieved.

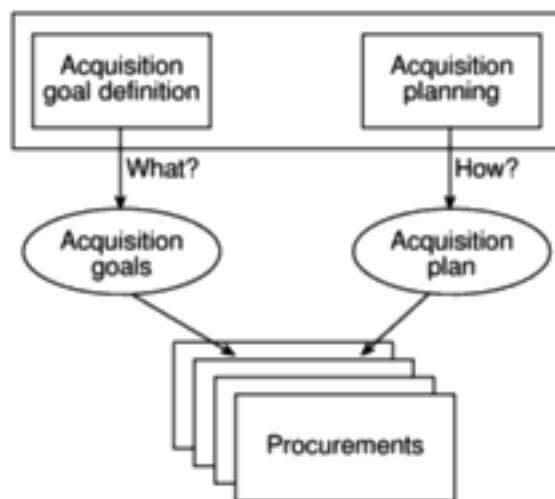


Figure C.4 The acquisition process is made up of procurements (contracts), each of which can cover one or more adaptations (projects).

The main steps in this process are shown in Figure C.5.

C.4 Acquisition goal definition

Define target domain

We have already suggested above that EM has a model where an IS development project is seen as an adaptation or re-engineering of an existing information system (and its embedded computer system where relevant). In the terminology employed by EM, the part of the business process that is to be changed is called the *target domain* and, as the first task in the acquisition process this needs to be

carefully delineated. One outcome of this could be the identification of a number of more manageable sub-domains.

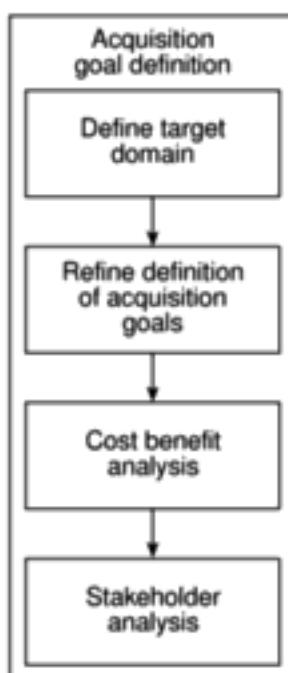


Figure C.5 Acquisition goal definition.

Refine definition of acquisition goals.

Deciding which parts of the current business are to be re-engineered might well lead to a better definition of what precisely the customer hopes to achieve and what systems they need to acquire.

Cost benefit analysis and stakeholder analysis

The refined definition of the acquisition goals can be used to estimate what the practical benefits and cost of the final system are likely to be and to pinpoint all those who are likely to be affected by the proposed changes.

C.5 Acquisition planning

Identify plan scenarios

The information gathered above will be used for acquisition planning. To a certain extent, this can go on in parallel with the goal definition. For example, consideration of what is involved in obtaining the components of the projected information system will give a better idea of its costs. In the light of this, it might be decided that the projected information system is not in fact financially viable. The requirements gathered by the acquisition goal definition can now be broken down into packages and the best order for the delivery of those packages can be judged. These details are called in EM terminology a *plan scenario*. This process is rather like Gilb's Incremental Delivery Plan in relation to intentional

The main steps in acquisition planning are listed in Figure C.6.

incremental delivery. As in that case, the order in which increments have to be delivered will be driven initially by technical constraints – for example, hardware might have to be installed before software can be loaded. Once these technical considerations have been satisfied, then customer preferences about which parts of the system should be delivered first can be taken into account.

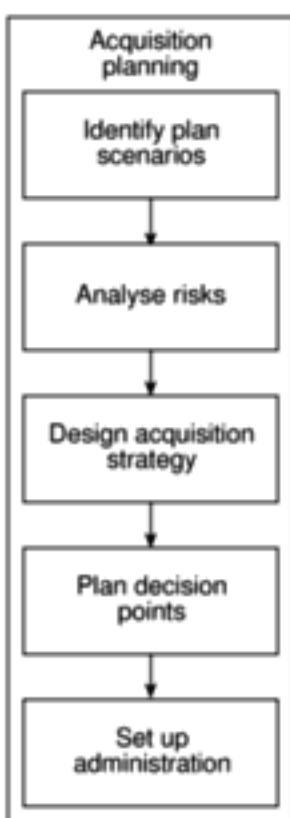


Figure C.6 Acquisition planning.

Analyse risks

Having produced one or more plan scenarios, these can be examined to see what risks they might involve. The EM approach to risk analysis and management is explored in greater detail in the section on adaptation planning. Needless to say, this risk analysis is a major influence on the acquisition strategy to be adopted.

The risks that are detected can be countered in a number of ways. One way is for the source of the risk to be removed. It could be, for example, that one risk derives from the lack of experience of the IS developers in a particular technology. This risk could be removed by using a technology with which they were familiar. Another way this particular risk could be overcome might be to contract work out to an organization that did have the required skills.

Design acquisition strategy

Important decisions will be recorded in the acquisition strategy document. They include the way that the acquisition programme is to be divided up into contracts and the order in which those contracts are to be placed and carried out. The EM

approach provides extensive guidelines about the grounds on which these decisions should be made.

Plan decision points

The main *decision points* in the acquisition process can now be plotted. These are the milestones where the customer will receive key deliverables and, on the basis of these, will make key decisions about the progress of the acquisition.

Set up administration

The final part of this initiation process is the setting up of the administrative and management structures in the customer organization to deal with the remainder of the process.

C.6 Procurement

After the planning processes described above, the acquisition process proper can begin. For each contract, there will be a procurement made up of two phases dealing with the *tendering process* and the *contract monitoring process*. These largely follow but also elaborate on the ISO 12207 acquisition process framework. The interested reader is directed to the EM document for further details.

ISO 12207 is dealt with in Appendix D.

C.7 Adaptation planning

An acquisition process as described above could involve several individual adaptations that might take place in sequence – see Figure C.3 above – or in parallel. For each of these adaptations, an adaptation plan will need to be formulated. The way in which this is built up follows a characteristic EM risk-driven approach.

Figure C.7 provides an outline of adaptation planning. Each of the steps in that outline will now be examined in turn. As an example, let us take a relatively small adaptation. As part of a wider acquisition, an organization needs to have user documentation for a new information system written in German for use by the German part of the organization. In this environment, the work practices vary from country to country and so a simple translation of the English language version would not be adequate. It is decided to contract this work out.

Risk Analysis

This is made up of the steps shown in Figure C.8. Firstly steps are taken to ensure that the initial and final IS states are adequately described and documented. In the example where German language documentation is to be produced, the question would be: ‘what documents are available upon which to base the German user document?’ A problem could be that at the moment the final shape of the system is still not completely known.



Figure C.7 Steps in Adaptation planning.

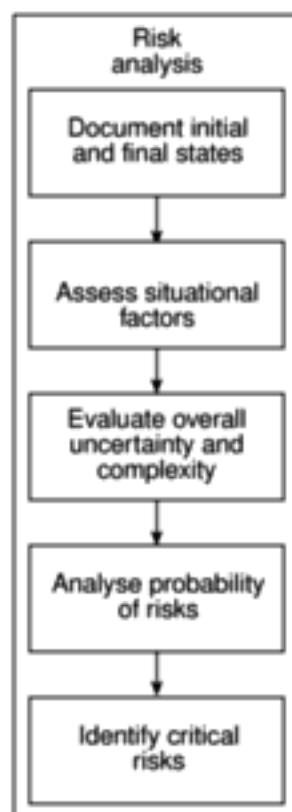


Figure C.8 The steps in risk analysis.

Once the start and end points of the adaptation have been clarified, the *situational factors* in the problem area have to be assessed. These are the factors that could make the adaptation difficult to carry out. An example in our German language documentation case is that the users of the information system belong to a number of different groups of staff who are each using the system in a rather different way. Hence what might be good documentation for one set of users might not suit the needs of another group.

To assist in the identification of risk, EM identifies two broad categories of risk: those to do with *complexity* and those to do with *uncertainty*. Some activities are inherently risky simply because they are large and complex. On the other hand, the source of risk might not be the level of detail involved but the uncertainty of those details. For example, it could be that the application is in an area where the requirements are constantly changing because of environmental impacts. In the case of the German version of the user documentation, for example, it could be that the system that the documentation is trying to describe is itself subject to frequent change.

These elements of uncertainty and complexity could be inherent in the system to be delivered or adapted (that is, the *target domain*) or in the processes that are going to be carried out to deliver or adapt the system (that is, the *project domain*).

Within the project domain, the uncertainty and complexity factors can be further categorized as being related to the nature of the tasks that need to be carried out, the *structure* of the project, the *actors* involved, and the *technology* to be employed.

Each risk that is identified is given a rating as 'high', 'medium' or 'low'. EM provides a list of the commonly occurring risks under each of the categories and sub-categories mentioned above.

EM now specifies a step where the analyst in some way summarizes these risk factors in order to obtain some general judgement of the overall complexity and uncertainty of the adaptation. Unfortunately, EM does not provide guidance on how this is to be done. It is also not clear why this is done at this point and not after the final two steps in the risk analysis procedure that assess the probability of each risk's occurring and its possible impact, thus identifying the critical risks in the adaptation.

Design of an adaptation strategy

The steps in this part of the adaptation planning process are outlined in Figure C.9. Having identified the potential risks in the adaptation, counter-measures can now be contemplated. EM provides some guidelines as to the kind of measures to be used against certain types of threat, although there is no claim that the advice by any means exhausts this subject.

With the example of producing German language user documentation, a risk was identified that the base system could change. This could be interpreted as an example of the generic risk 'evolving requirements'. EM suggests as possible counter-measures the implementation of a stringent change control procedure to discourage unnecessary changes, and measures to make sure that delivered products are easy to change. The latter suggestion could be implemented by, for example, having supporting documentation that cross-referenced the user documentation to the system specification so that the effect of changes in the base system could be quickly traced through to user documentation. The user documentation might also be structured to make use of appendixes for details such as error messages, which are likely to change.

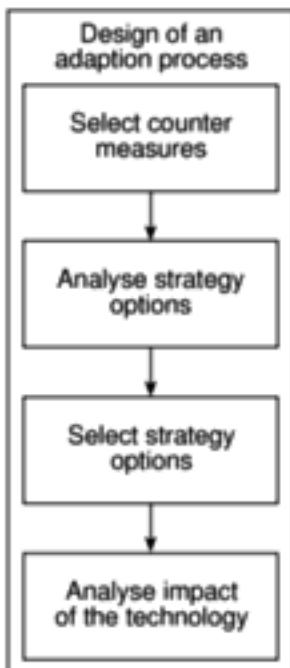


Figure C.9 Design of an adaptation strategy.

The risks to an adaptation can also be dealt with at a higher level by considering the overall approach to the adaptation project.

This will influence:

- the *description* approach;
- the *construction* approach;
- the *installation* approach.

The description approach is concerned with the method to be used to describe the systems involved. This is divided by EM into a *cognitive* approach and a *social* approach.

One of the two alternative cognitive modes is *analytical* where information about the system is abstracted into simplified views that focus on certain aspects of the system to the exclusion of others. For example, in SSADM, the Logical Data Structure (LDS) is an abstraction of the data needed by a system. On the other hand, the *experimental* mode, as its name implies, elicits information by using experiments to learn about the system. The use of prototypes is a central technique in this mode. The analytical and experimental modes are not mutually exclusive.

The social approach is the way that the developers and users work together. This can be either in an *expert-driven* or a *participatory* mode. In the expert-driven mode, systems analysts, or the equivalent, will fact-find among the users and then go away and, on their own, produce the system description. In the participatory mode, the descriptions are produced in a joint effort between developers and users – Joint Applications Development exemplifies this approach.

Each of the cognitive approaches can be used in conjunction with either of the social approaches.

In common with ISO 12207, EM offers advice on whether a *one-shot*, *incremental*, or *evolutionary* approach would be best. The distinctive feature of EM is that it distinguishes between *construction* and *installation*. It allows, for example, for a system to be built in increments but to be installed in one shot – or vice versa.

As well as the one-shot, incremental and evolutionary options in relation to construction and installation, EM provides guidance on whether, geographically, installation should be carried out in all locations in one step, or in stages where groups of locations are dealt with in turn.

The general *heuristics* provided by EM about the most suitable approach to description, construction and installation are shown in Table C.1 and Table C.2.

Table C.1 General heuristics of suitability of installation/construction approaches

Situational factor			Installation/construction approach		
Schedules	Complexity	Uncertainty	one-shot	incremental	evolutionary
normal	simple	certain	✓		
		uncertain		✓	
	complex	certain		✓	
		uncertain			✓
tight	simple	certain	✓	✓	
		uncertain			✓
	complex	certain	✓		
		uncertain			✓

'Heuristic' is defined as 'furthering investigation but otherwise unproved or unjustified' in the Longman Concise English Dictionary

Table C.2 General heuristics on the suitability of description

	Expert-driven	Participatory
Analytical	Situation is <i>complex</i> but information/business processes are relatively <i>straight-forward</i>	Information/business processes are relatively <i>complex</i>
Experimental	Situation is <i>uncertain</i> but participatory approach unsuitable	Situation is <i>uncertain</i> but personnel are experienced and time-scales are adequate

EM suggests that these options be considered in the following order:

1. installation – system coverage (that is, one-shot, incremental, evolutionary);
2. installation – geographical coverage;

3. construction;
4. description.

These ‘first-cut’ strategy options might be revised in the light of the more detailed heuristics that are provided by EM. For example, although the general heuristic guidance might point to an analytical/expert-driven approach, because of the importance of winning and maintaining the support of the users for the new system, you might decide to plump for a more participatory approach.

While the adoption of a particular approach to the adaptation could have reduced some risks, it is possible that it can generate completely new ones – the impact of the strategy has to be scrutinized to remove this possibility.

Project control

Finally, advice is offered about how the project would be most appropriately controlled. The project control options to be considered here relate to the frequency of decision points, the degree of formality needed in the control process and the question of customer responsibilities – for example, who should have the responsibility for user training: the supplier of the adaptation or the customer?

These decisions have to be made in respect of control of three different kinds: development control, which is concerned with the progress of the project, quality control, and configuration control.

Plan decision points

The selection of the approaches will have a major influence on the positioning of decision points (or ‘milestones’) in the project lifecycle. For example, decision points would seem to fall naturally at the end of each increment where an incremental approach is adopted, and at the end of each iteration with an evolutionary approach. The nature of the initial and final IS states will also influence the placement of decision points – the further apart they are in the system lifecycle, the more decision points will need to be planned.

For each decision point, the deliverables that are needed for that milestone and the decisions that will have to be taken at that point will have to be recorded. These deliverables are defined generically in EM, but they will have to be mapped to the types of product created by the suppliers’ chosen methodology. This is discussed as a separate topic below.

The final guidance proffered by EM consists of the criteria that can be applied to assess the appropriateness of the proposed adaptation plan.

C.8 Method bridging

The outcome of these EM activities is an *adaptation plan*, which identifies decision points and the deliverables that supply the information upon which the decisions are to be taken. These decision points and deliverables have to be reconciled with the development methodology that is proposed.

EM recognizes six views of an information system.

1. The *business information view*, which defines the data that is used by the business and how it changes over time.
2. The *business process view*, which identifies the business processes that are carried out and the rules to which they conform, and how they relate to the goals of the business.
3. The *work practice view*, which describes how people actually execute the business processes, using the business information that is to hand.
4. The *computer system data view*, which describes how the business information requirements are actually implemented in the computer system.
5. The *computer system function view*, which identifies the transactions that users can carry out using the computer system.
6. The *computer system architecture view*, which describes the actual way in which the software is segmented and structured.

For each of these views, there are identified a number of *properties* that are essential pieces of information about the view. In the case of the business information view, for instance, the items of information held, their value ranges and the relationships among them are all properties.

For the methodology it is proposed to use, say Information Engineering, the particular products that document those properties should now be identified. It could well be that a methodology does not provide some of the information that is required (it appears, for example, that some of the modifications introduced in the latest version of SSADM have been introduced to remedy deficiencies identified by EM). On the other hand, it might be that not all products of a methodology are required by a particular adaptation – the delivery plan with its list of decision points and associated deliverable profiles will show which ones are needed.

C.9 Conclusions

It will be interesting to see how widespread the adoption of Euromethod will be. It could be that, ironically, the name itself will restrict its adoption as non-European developers might automatically ignore it! It was suggested at one point that an alternative name, which was less parochial, such as 'EMpathy' might be better.

Another disadvantage of the method is the sheer size of the documentation and the obscurity of some of its terminology. Standards documents are never a riveting read but EM particularly suffers in this respect.

Despite this, we have found that there is a lot of worthwhile material in EM. Its strength is that it attempts to give practical advice about how the decisions are to be taken, rather than just stating what the decisions are. We would look forward to the development of a more user-friendly version as time goes on.

Appendix D

ISO 12207 – an overview

D.1 Introduction

Both PRINCE 2 and BS 6079 originated in the United Kingdom and are designed for any type of project. ISO 12207, differs from these in that, firstly, it is international in standing and secondly that it relates specifically to software development.

Broadly speaking, ISO 12207 has as the prime areas of its concern the documentation (or ‘software life cycle data’ as it calls it) created and used by a software development project and the processes that, during development, will use and update software life cycle data.

It is stressed that this is only an overview and those using the standard in earnest need to refer to the actual standard itself.

D.2 The ISO 12207 approach to software life cycle data

‘Documentation’ is an issue that is difficult and important, but at the same time rather unexciting. Software developers and users often complain about the lack of documentation, but when it is available it often remains unread. There could be some justification for this reluctance to read documentation as it might well not be up-to-date.

One way to look at a software development project is as an information system in its own right. The project is made up of activities, each of which needs to pass information to and from other activities. As with any conventional information system, there needs to be a common database that can be updated and accessed as required. A key factor in the relative success and failure of the project is clearly going to be the effectiveness of this information system.

However, inappropriate documentation can actually be an obstacle to effective working. ISO 12207 focuses attention on the characteristics of good documentation by firstly defining the purpose of good software life cycle data. This data:

- records information about *software products*;
- helps make the product *usable* and *maintainable*;
- defines *processes*;
- *communicates* information;
- records *history*;

We have chosen to use the term ‘documentation’ partly to make indexing easier. ‘Documentation’ could imply paper-based information but, of course, in practice, it could be held in an electronic form.

- provides *evidence*.

The standard lists the characteristics of good documentation as being:

- unambiguous;
- complete;
- verifiable;
- consistent – that is, there are no contradictions within it;
- modifiable;
- traceable – the components from which it is derived are easily identifiable;
- presentable – this means that it is easy to access and view

The standard recognizes the following generic types of data held by a project:

- requirements – what the system is expected to do;
- design – including details of its structure;
- testing – including test strategy and criteria, test cases and results;
- configuration;
- user – including user manuals;
- management;
- quality – including quality plans and procedures.

IEEE stands for the
'Institute of Electrical and
Electronics Engineers',
the prestigious US-based
organization that has
played a key role in setting
standards.

The ISO 12207 standard does not specify precise formats for this documentation but the IEEE, for example, has produced a cross-reference indicating the relevant IEEE standards for each type of document.

D.3 The ISO 12207 approach to software life cycle processes

The processes that, either as a central objective or as a by-product, generate or update life cycle data are particularly the concern of the standard.

It will be recalled from Appendix A and Appendix B that BS 6079 differed from PRINCE 2 in defining, in theory at least, a project as starting with a concept and ending, after the required system had been both built and operated, with its decommissioning. In practice with BS 6079, however, the detailed procedures suggested for project planning and control focused on the development phase.

ISO 12207 identifies five distinct processes:

- acquisition;
- supply;
- operation;

- maintenance;
- development.

Clearly, maintenance and operation are processes that genuinely belong to the post-implementation phase of a conventional development project. One justification for the inclusion of these post-implementation activities is that sometimes a customer will contract a supplier both to develop a system and operate it on the customer's behalf. Similarly a software house could be responsible contractually both for the construction of a software-based system and its maintenance after installation. Our attention here is given primarily to the acquisition, supply and development processes. It must be stressed once again that this appendix is intended merely to give an overview. Those intending to use the standard 'in anger' must obtain and study the full document.

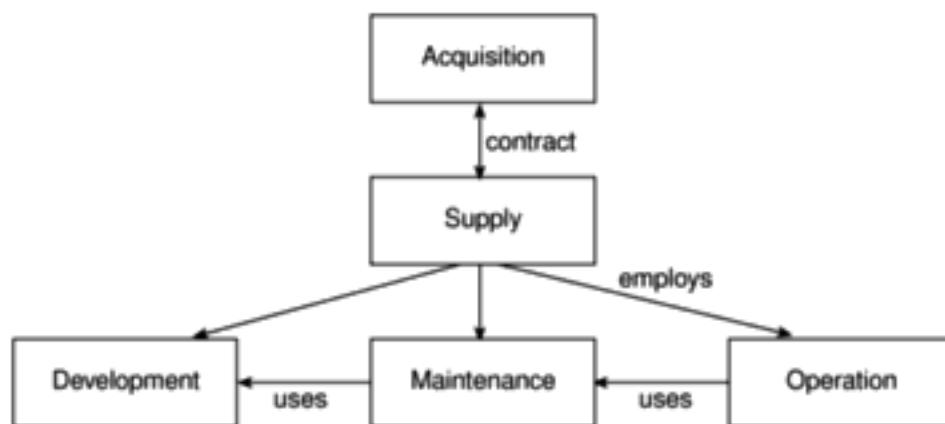


Figure D.1 ISO 12207 processes.

The acquisition process is the set of procedures that a customer for software (or 'acquirer' in ISO 12207 terminology) will go through in order to obtain that software from an external source. The supply process is the opposite side and is the set of procedures that the supplier should adopt in order to satisfy the acquirer's needs. The supplier might use an existing piece of software or might develop new software, in which case they would need to invoke the development process.

Just as in the case of BS 6079, some supporting processes can be identified. These are listed in Table D.1.

We will now deal briefly with the acquisition, supply, and development processes in turn.

D.4 The acquisition process

A previous discussion in the chapter on contract management has already mentioned ISO 12207 in this context and so we will avoid going into excessive detail here.

Table D.1 ISO 12207 Supporting and organizational processes

<i>Supporting processes</i>	<i>Organizational processes</i>
documentation	management
configuration management	infrastructure
quality assurance	improvement
verification	training
validation	
joint review	
audit	
problem resolution	

Figure D.2 portrays the main activities that compose the acquisition process. The *initiation* activity starts with the description of the ‘concept’ that the acquirer wishes to make real, or the need that the acquirer wishes to satisfy. The requirements of the system then need to be defined by the acquirer. In fact, the acquiring organization could employ an external source to do this for them, but they would still have the responsibility of approving these requirements.

ISO 12207 distinguishes between system requirements and the software requirements that now have to be analysed and documented – software requirements relate to the distinct software components that will make up the delivered system. Once this has been done, a decision needs to be taken about the best way to acquire the software, for example, whether to make or buy. This is analogous to the ‘defining the project approach’ (SU4) activity in PRINCE 2.

Having made this decision, the acquirer is now in a position to prepare the acquisition plan, detailing the steps needed to acquire the software, taking account of such matters as who is to be responsible, for example, for any maintenance and support. Any inherent risks need to be considered. It is also important that at this point the criteria for final system acceptance, and the methods by which compliance is to be evaluated, be defined and recorded.

Request for proposal (RFP) The groundwork has now been done for the production of the ‘request for proposal’ document. This should include sections on the topics listed in Table D.2.

An important activity in the context of ISO 12207 is the tailoring of the standard for a particular project. In some places in the standard, it is stated that certain process requirements should be ‘as specified in the contract’. The precise requirements to be included in the current contract now need to be identified. Depending on the type of product or project, it could be agreed that certain ISO 12207 requirements are to be dropped as not appropriate, while in other cases there could be process requirements above and beyond those in ISO 12207 that need to be documented.

Contract preparation and update Before you can have a contract, you must have a supplier with whom to have the contract. With this in mind, the criteria to be used in selecting the supplier and the method by which the compliance by

Another name for ‘request for proposal’ is ‘invitation to tender’

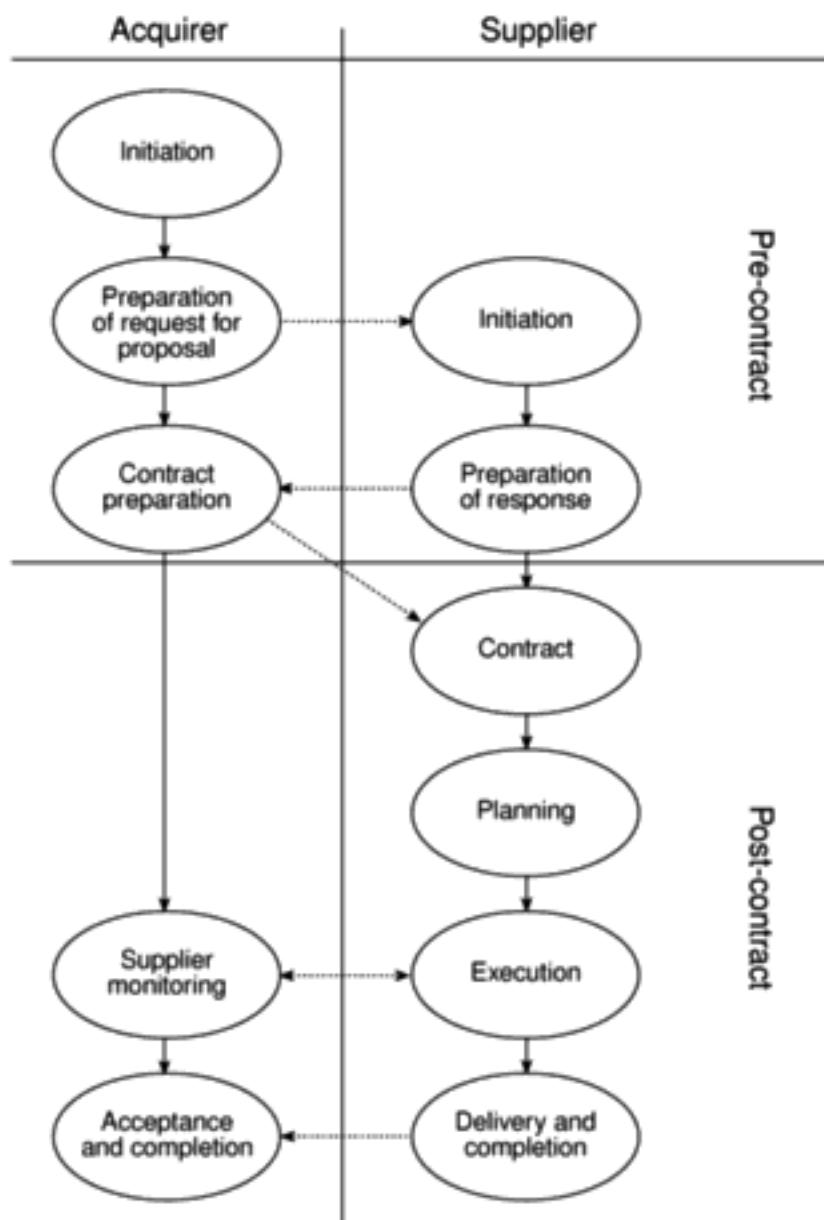


Figure D.2 *Interaction of acquisition and supply processes.*

Table D.2 *Topics in a request for proposal document*

Request for proposal – contents

system requirements

scope statement

instructions for bidders

list of software products

control of sub-contracts

technical constraints (e.g. the target environment)

potential suppliers with the criteria can be judged have to be set down. Once the preferred supplier has been selected, the final form of the contract between the supplier and acquirer can be negotiated. This often involves some adjustments to the way in which ISO 12207 is to be tailored.

Monitor supplier This will be done using some of the supporting processes listed in Table D.1, namely joint reviews, audit, verification and validation.

Accept completed contract When the supplier finally delivers the product, the acquirer will conduct acceptance tests and if the specified acceptance criteria are satisfied, the completed software can be signed-off as completed.

D.5 The supply process

This process mirrors the acquisition process, but documents the activities that a supplier would need to undertake in response to the request of a supplier. Figure D.2 outlines the main activities involved. There will be occasions where the sequence of activities is not that shown and yet other occasions when some of the activities will have to be iterated a number of times.

Initiation The process is started when a potential supplier receives an RFP from an acquirer and the supplier decides to bid for the work.

Preparation of a response The supplier, after consulting people with various types of expertise, now prepares a response. This should include proposals about how ISO 12207 is to be tailored for the project in view.

Contract If all goes well, the supplier's proposal will make the right impression and lead to acceptance by the acquirer. The details of the contract are then negotiated and signed.

Planning The supplier can now draw up a detailed plan of how the work is to be done. The starting point for this will be the requirements as laid down in the RFP. You would normally expect this to include the life cycle approach to be applied by the supplier, as this will influence the points during the project at which consultation between supplier and acquirer is to take place. If the life cycle has not been stipulated, then it should be selected now as a basis for devising the plan. It will have been noted that earlier the acquirer might have considered, as part of the acquisition process, the options of 'make' versus 'buy' and also whether to use in-house or external sources. ISO 12207 now makes provision for the supplier to make a similar choice, and having done this, to develop a plan accordingly. For example, a contractor who is primarily a hardware specialist might use a software house to write the software required as part of a contract.

The general format of the plan is shown in Table D.3.

Execution and control The plan can now be executed. Depending on whether the contract is for systems development, operation, or maintenance, the ISO 12207

Table D.3 *Contents of a plan*

<i>Topics to be covered by a plan</i>
project organizational structure
engineering (i.e. development) environment: including facilities, standards, procedures and tools
work breakdown structure
management of quality characteristics
management of safety, security and other critical requirements
sub-contractor management
verification and validation
acquirer involvement
user involvement
risk management
security policy – ‘need-to-know’ and ‘access-to-information’ rules
arrangements for obtaining any regulatory approvals
scheduling, tracking and reporting
training

process stipulated for development, operation, or maintenance would now be undertaken.

During the execution of the plan, the standard expects the supplier to monitor and control progress and product quality, and to have a mechanism for recording, analysing and resolving the problems that occur. The supplier will also be responsible for passing on requirements that accurately reflect those of the acquirer to any sub-contractors and for ensuring the compliance of sub-contractors with those requirements. The supplier also needs to co-operate fully with any independent verification and validation processes that were laid down in the contract.

Review and evaluation The provisions here are mainly to ensure that the supplier allows the acquirer access to the information needed to review the progress of the project, although the precise extent to which the acquirer has a reviewing role and access to supplier documentation has to be specified in the contract.

Delivery and completion Attention is required in any management plans to the way products are to be delivered and to how any required post-delivery support is to be provided.

D.6 The development process

ISO 12207 sees the development process as being made up of the activities in Figure D.3. Most of the terms used will be familiar to software developers. Some

activities address the *system* as a whole – for example, the analysis of user requirements and the overall design of the interacting components that together should meet those requirements (architecture design in Figure D.3). Others relate to *software*, which in practice means activities that are repeated for each *software item* in the system. The standard identifies a hierarchy of *software items* in a system that are made up of *components* that, in turn, are made up of *software units*. The software unit is the basic unit of code, but the precise definition of components and items will depend on the circumstances of a particular project. At the same time that these activities are starting, progressing and terminating in the planned sequence in a project, there should be the continuous background operation of supporting activities, such as configuration management and problem resolution – this is represented in Figure D.3 by the ‘process implementation’ stream.

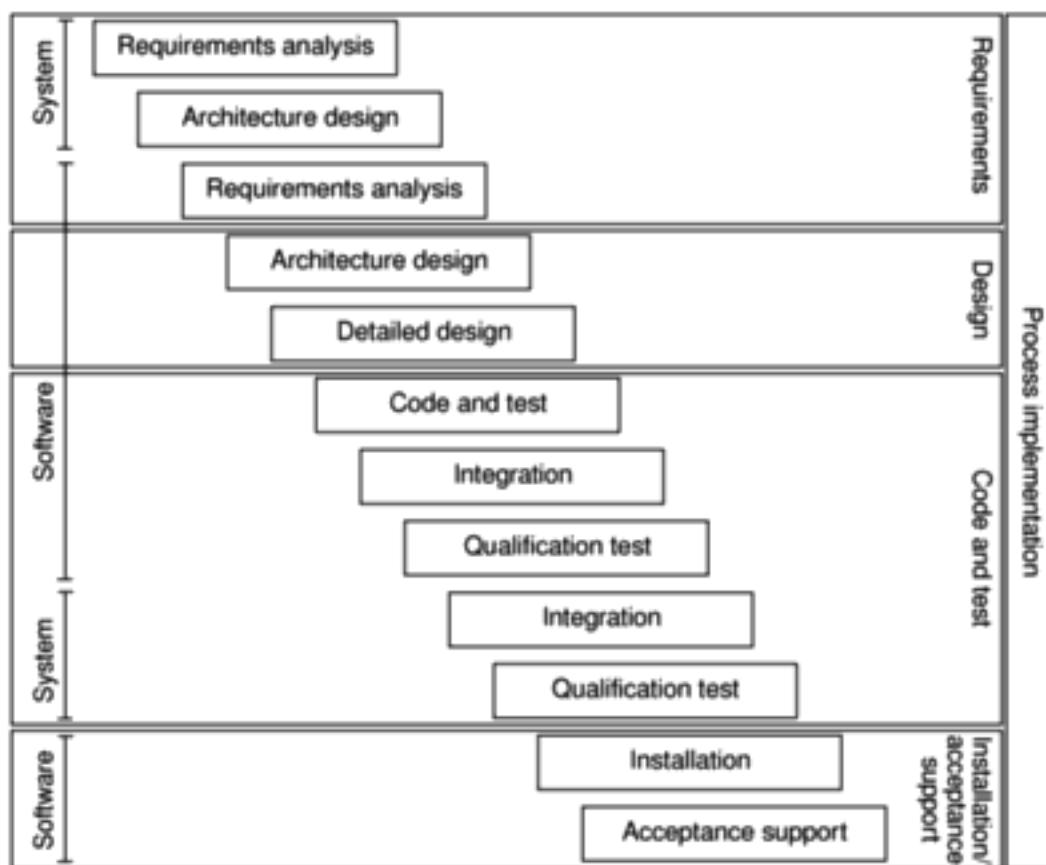


Figure D.3 ISO 12207 development software life cycle.

‘Once-through’ in IS 12207 equates to one-shot in Euromethod.

The structure in Figure D.3 implies a ‘waterfall’, or in ISO 12207 terms a ‘once-through’ approach. Figure D.4 and Figure D.5 illustrate how these activities can be shuffled to fit an incremental or evolutionary approach to a project. As in the case of Euromethod – see Appendix C – some rather simplistic guidelines are given to help the planner decide which approach to adopt – for example, if the acquirer wants all the system capabilities in the first delivery, an incremental approach would not be advisable!

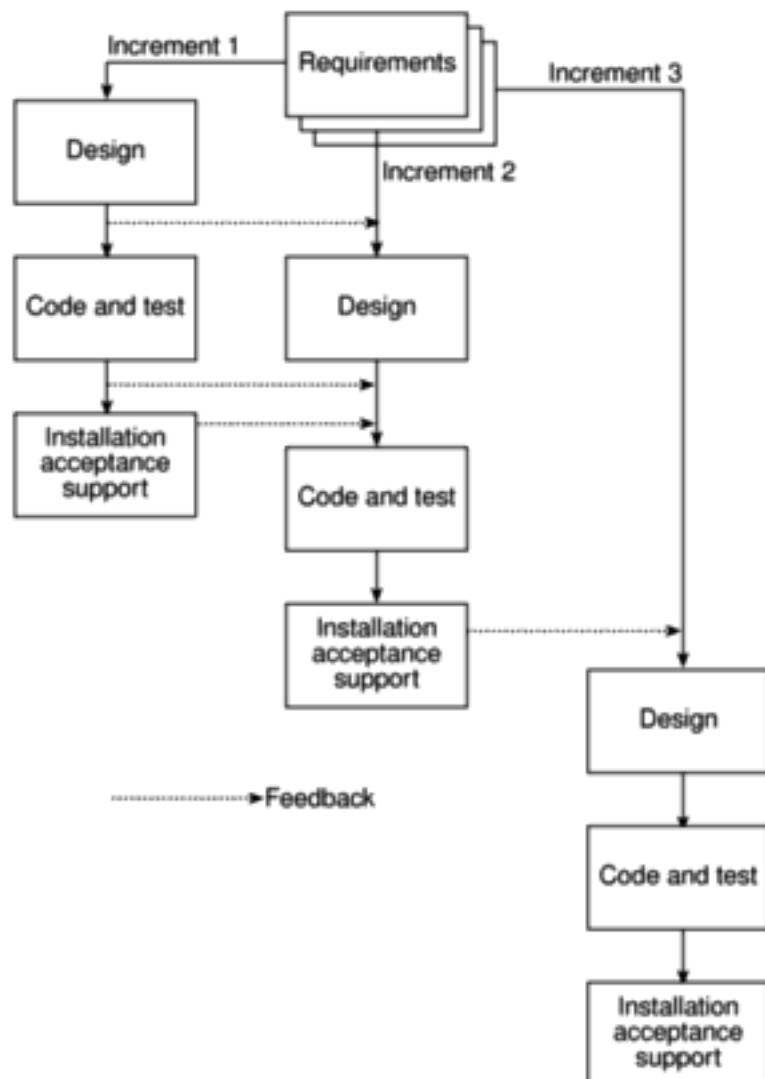


Figure D.4 Using ISO 12207 activities in an incremental project.

Guidance as to what should be done for each activity is given and the reader who needs this level of detail really should read the standard itself. The general picture that emerges is that for each activity there should be the following pattern:

- plan;
- do;
- document;
- evaluate.

It is to be hoped that in general the doing and the documenting are going on in parallel, rather than the documentation following the doing as an easily forgotten afterthought. A prime objective of ISO 12207 is to ensure that appropriate and correct data is recorded.

The evaluation procedure following the carrying out of an activity is usually done by means of a joint review, or, where testing has taken place, through audits. Reviews can be at a management level where the project is monitored or can be at

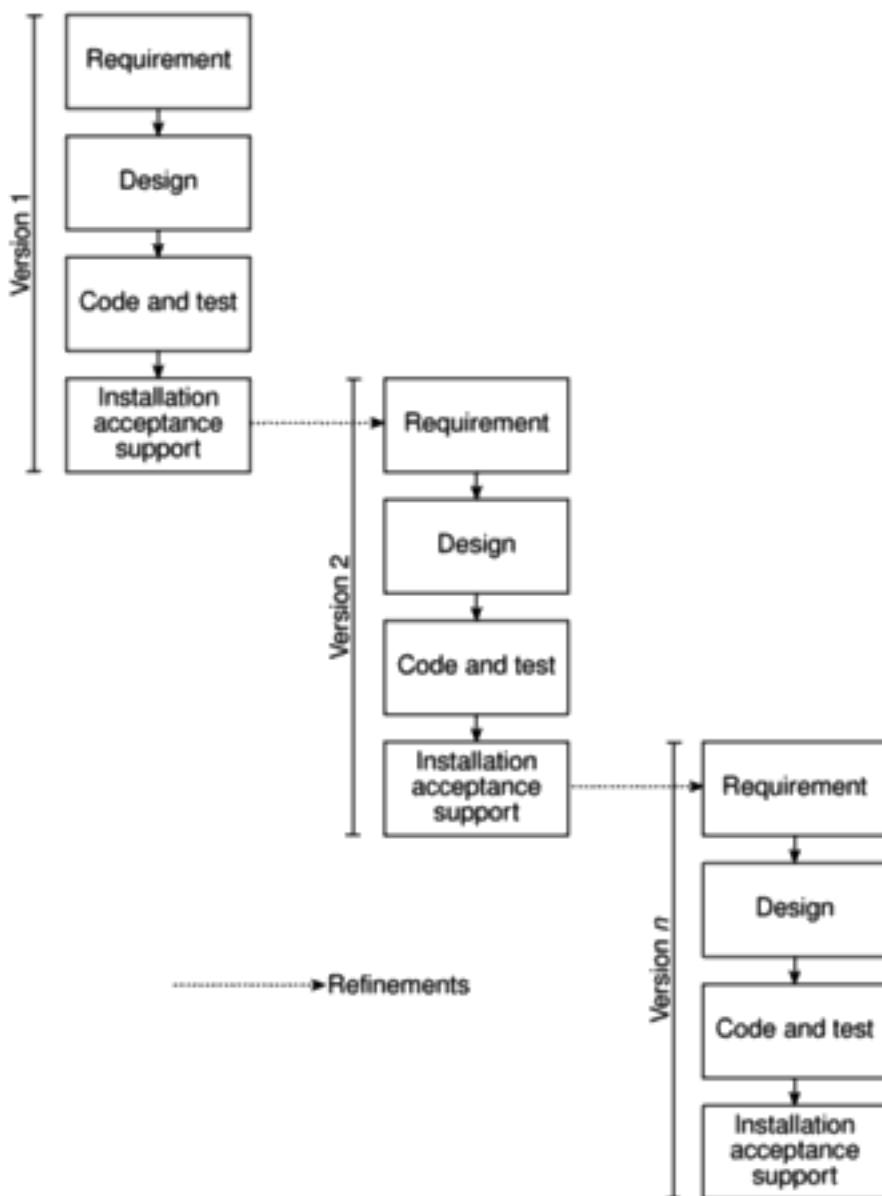


Figure D.5 Using ISO 12207 activities in an evolutionary project.

a technical level where the appropriateness of a particular software product is examined. The technical reviews will look at such considerations as:

- is the product complete – or are some parts missing?
- does it comply with the standards and the specification?
- have all required changes been carried out?
- is it on schedule?
- will it be an acceptable basis for the next activity?
- do the processes used follow those in the plans, schedules, standards and guidelines for this project?

In addition, some concerns will relate to the products created by specific activities as Table D.4 shows. The emphasis in many cases on the 'feasibility of operation and maintenance' is worth noting. The software does not only have to meet the acquirer's functional requirements, it needs to work in the planned operating environment and be easy to modify to meet any new requirements. Making these requirements explicit during development is invaluable.

Operation and maintenance processes

As noted earlier, the ISO 12207 standard also prescribes general processes for operation and maintenance. The interested reader is directed to the standard itself for further details of these.

Table D.4 IS 12207 generic acceptance criteria

	Traceability to source documents	Consistency with previous products	Internal consistency	Testability	Test coverage	Conformance to expected results	Appropriate standards and methods	Feasibility of next activity	Operational feasibility	Maintainability
System requirements analysis	✓	✓		✓			✓	✓	✓	✓
System architecture design	✓	✓				✓	✓	✓	✓	✓
Software requirements analysis	✓	✓	✓	✓			✓	✓	✓	✓
Software architecture design	✓	✓	✓			✓	✓	✓	✓	✓
Software detailed design	✓	✓	✓	✓		✓		✓	✓	✓
Software code and test	✓	✓			✓	✓	✓	✓	✓	✓
Software integration	✓	✓	✓		✓	✓	✓	✓	✓	✓
Software qualification test					✓	✓	✓	✓	✓	✓
System integration					✓	✓	✓	✓	✓	✓
System qualification test					✓	✓		✓	✓	

Appendix E

Project Management Bodies of Knowledge

E.1 Introduction

The standards and guidelines we have looked at in the preceding appendixes have, to a greater or lesser extent, concentrated on procedure – the actions to be taken in specified circumstances. As we have noted before, one way of looking at the software development process is as an information system in its own right: processes take information from various sources, then manipulate and process it to create new information products that are used by other processes. It is clearly in the interests of the managers of a project, especially a larger one, to ensure that this is done in an accurate and structured way and standards such as ISO 12207 and PRINCE 2 focus on this aspect.

Correct procedure needs to be supported by professional competence. Administratively, a procedure can be carried out correctly so that the correct documents have been completed recording the decisions made, but if the project manager is not competent, the decision that has been conscientiously recorded can still be misjudged and damaging. Competence will stem partly from the innate ability of practitioners, their use of past experience and formal learning. In this Appendix, we are going to look at some attempts that have been made to define what knowledge and skills the good project manager should have. In the examples we will look at, this has been done in order to form the basis for some kind of practitioner certification. This will be of some assistance to readers who are considering acquiring a formal qualification in project management.

The reader should be warned that the information given in this appendix is very much subject to change. We know of at least two cases (APM and ISEB) where particular bodies of knowledge are under review and will certainly be revised. Readers are directed to our website at <http://www.mcgraw-hill.co.uk/hughes> where we will endeavour to keep details up to date.

As noted in Appendix A, it is possible to be assessed as a PRINCE 2 practitioner – we have treated this as a specialized qualification in a particular technique rather than a general PM qualification.

E.2 Project Management Institute

William R. Duncan's
'Developing a project-management body-of-knowledge: the US Project Management Institute's approach 1983-94' in the *International Journal of Project Management* 13(2) pp89-94.

The Project Management Institute (PMI) was founded in 1969 in the United States and currently has a membership in the region of 34,000 members world-wide. In 1983 it published a 'Project Management Body of Knowledge' (PMBOK). Since then the PMBOK has been extensively revised and expanded – it now totals over 150 pages with appendixes – so that the 1996 release, which has the more realistic title of 'A guide to the Project Management Body of Knowledge' is a mature product. One enormous advantage that it possesses is that it is possible to download it over the World Wide Web from the PMI site.

The document is based on the concept of there being bodies of knowledge upon which various professions, such as medicine or law, base their practice. The PMBOK is an attempt to delineate such a body of knowledge for the project management professional. Clearly this could be an enormous undertaking and the PMI has tried to make it more manageable by attempting only that subset which is generally accepted. Even with this restriction, the potential scope is extensive and the fact that the document cannot hope to be all-inclusive is reflected in the word 'guide' in the title.

The objectives of the PMI in producing the PMBOK have been to:

- identify generally accepted project management practice;
- produce a basic reference document;
- identify a common set of terms;
- act as a basis for training and accreditation.

The compilers of the body of knowledge have therefore had to draw the body of 'project management' with some care. On the one hand, there are general management principles and practice, some aspects of which have a bearing on projects while others do not. On the other hand, there are other aspects of project management which are only relevant to some technical applications. An example of this would be software project management where some material is peculiar to software development, for example the use of function points for effort estimation.

The starting point of PMBOK is a definition of a project as 'a temporary endeavour undertaken to create a unique product or service' and of project management as 'the application of knowledge, skills, tools and techniques to project activities in order to meet or exceed stakeholder needs and expectations from a project'. This last definition seems to us to be a little too widely drawn as one can imagine technical activities that could fall within this definition.

The PMBOK is organized into nine key knowledge areas, which have further subdivisions as shown in Table E.1. Notice how every topic section name is prefixed by 'project' to make clear that, for example, 'human resources management' does not cover all the topics that might come under that heading – only those that relate specifically to project management

The compilers of PMBOK have experienced the same problem that we experienced in structuring this book: the same subject topic can apply to several

Table E.1 PMBOK contents

<i>Section</i>	<i>Subsection</i>
Project integration management	Project plan development Project plan execution Overall change control
Project scope management	Initiation Scope planning Scope definition Scope verification Scope change control
Project time management	Activity definition Activity sequencing Activity duration estimating Schedule development Schedule control
Project cost management	Resource planning Cost estimating Cost budgeting Cost control
Project quality management	Quality planning Quality assurance Quality control
Project human resource management	Organizational planning Staff acquisition Team development
Project communications management	Communications planning Information distribution Performance reporting Administrative closure
Project risk management	Risk identification Risk quantification Risk response development Risk response control
Project procurement management	Procurement planning Solicitation planning Solicitation Source selection Contract administration Contract close-out

points of the project life cycle. This means that the time sequence of concerns that a project manager has while planning and executing a project does not map neatly onto the organization of the subject matter. For example, risk management as a set of techniques will be exercised in several places throughout the progress of the project.

The problem has been tackled in PMBOK by identifying a set of processes for each phase of a project. A 'process' is defined as 'a set of actions bringing about a result'. Five groups of process have been identified: *initiating; planning; executing; controlling; closing*.

Within each group there are *core* processes. These are often inter-connected so that the output from one process is input by others. In common with most of the approaches that have been discussed in previous appendixes, there are a number of supporting processes (or *facilitating* processes in PMBOK terminology) such as quality planning and risk identification.

For each of the processes, PMBOK defines inputs, techniques that may be used and outputs.

This might seem to be similar to the way PRINCE 2 is organized, but such a comparison would be misleading, as the PMBOK describes the components of processes at a much higher and more abstract level.

In our view, PMBOK represents a remarkable achievement and has the great merit of being freely available via the PMI Web site. Inevitably, it is possible to disagree over some of the details, but that would be the case with any 'body of knowledge'. Our only real reservation (and perhaps this is because we come from an academic background) is the lack of supporting references to the authoritative works upon which a body of knowledge is based. This seems to be particularly a shame as the document does refer to itself as a 'guide' which, it might have been assumed, would mean that it directs readers to relevant material.

The PMI administer a certification examination (which it is possible to sit, not only in the US, but also in the UK). Once the written examination has been passed, candidates may apply to be assessed for full 'Project Management Professional' (PMP) status. However in 1998 it was reported that because of the strict practical experience requirements and the stringent assessment process, the growth in members achieving full PMP status has not been high compared to those taking the examination.

The address of the PMI in the United States is:
Project Management Institute
130 South State Road
Upper Darby
PA 19082
USA

The PMI have a website at <http://www.pmi.org>. There are some chapters of the PMI outside the USA, including in the UK, but local contact points are best obtained from the PMI centre (or our website at <http://www.mcgraw-hill.co.uk/hughes>).

Mark Becker, 'The time of the project manager',
Project Manager Today,
January 1998.

E.3 Australian Institute of Project Management

The roots of the Australian Institute of Project Management go back to the establishment of a Project Managers' Forum in Sydney, New South Wales, in 1976. In 1989, the organization was transformed into the Australian Institute of Project Management which, among other activities to support the professionalism of project management has undertaken the certification of competent project managers and the accreditation of project management courses.

See Alan Stretton:
'Australian competency standards' in *International Journal of Project Management*, 13(2) pp119-123.

The organization has developed sets of competency standards to support the certification of project managers which are recognized as the Australian national standards. These are generic so that they are not necessarily appropriate for direct use in all industries. An industrial sector or individual enterprise may establish their own standards suitable for their own environment, but these must be consistent with and as least as rigorous as the generic standard.

The AIPM has adopted a definition of a project as 'a unique set of inter-related activities, with defined start and finish times, designed to achieve a common objective'. Project management is defined as 'the integration of project activity through the project life cycle to achieve the delivery of a defined product or service within prescribed constraints of time, budget, scope and quality'.

The nine functions where competence needs to be shown are:

1. integration of project activities;
2. project scope management;
3. project quality management;
4. project time management;
5. project cost management;
6. project risk management;
7. project human resources management;
8. project contracting/procurement management;
9. project communications management.

The Australian Institute of Project Management are the custodians of the *Australian National Competency Standards for Project Management*. This has a structure similar to the NVQ/SVQ qualification in the United Kingdom (described later in this appendix).

The documentation for these standards is available via the world wide web and is of outstanding quality.

Enquiries can be addressed to:

The Secretary

Australian Institute of Project Management

PO Box 83

CAMPBELL ACT 2600 AUSTRALIA

E.4 Association for Project Management

The UK-based Association of Project Managers have also produced a 'Body of Knowledge'. Although the APM now sets and administers examinations in project management, at the time that this body of knowledge was formulated, their certification process was based, not on an unseen written examination but on assessing the practitioner's competence. The emphasis of the APM body of knowledge is therefore on defining competencies rather than knowledge areas as such.

To this end the APM has categorized projects into four broad levels in terms of scale and complexity:

- **level 1** – an in-house project involving a single disciplinary team;
- **level 2** – an in-house project involving a multi-disciplinary team;
- **level 3** – a multi-company multi-disciplinary project;
- **level 4** – a multi-country multi-company multi-disciplinary project.

Currently (1998) the APM is undertaking a revision of this body of knowledge.

The APM body of knowledge is a seven-page document divided into four general topic areas, which are broken down into forty sub-topics (see Table E.2). These sub-topics have been referred to as 'key competencies'. This can be a little confusing if the reader is expecting a 'competence' to be a definition of a particular task the practitioner should be competent to carry out (see the discussion of the NVQ qualification later). The APM list of competencies is actually a list of definitions. For example, the 'key competence' of 'project life cycle' is defined as 'the sequence of phases through which a project will pass from its conception to its completion'. It might be that the 'key competencies' are better regarded as 'topic areas'. In any case, associated with each of these 'key competencies' is a description of the knowledge and experience required by the practitioner and in most cases a list of references.

The APM now provides a three-hour examination assessing the candidate in the 40 professional competencies. Before being allowed to take the examination, the candidate has to submit an application form, a self-assessment form and a full curriculum vitae (résumé), which are assessed to establish the candidate's suitability. Evidence of three years' 'competent experience' has to be shown. Passing the examination allows those with more than five experience to become a Member of the APM and those with less than five years' experience to become an Associate.

Further information can be obtained from:

The Association for Project Management
85 Oxford Road
High Wycombe
Buckinghamshire
HP11 2DX
UK

Table E.2 APM 40 key competencies

<i>Section</i>	<i>Subsection</i>
1. Project management	1.1 Systems management 1.2 Programme management 1.3 Project management 1.4 Project life cycle 1.5 Project environment 1.6 Project strategy 1.7 Project appraisal 1.8 Project success/failure criteria 1.9 Integration 1.10 Systems and procedures 1.11 Close out 1.12 Post project appraisal
2. Organization and people	2.1 Organization design 2.2 Control and co-ordination 2.3 Communication 2.4 Leadership 2.5 Delegation 2.6 Team building 2.7 Conflict management 2.8 Negotiation 2.9 Management development
3. Processes and procedures	3.1 Work definition 3.2 Planning 3.3 Scheduling 3.4 Estimating 3.5 Cost control 3.6 Performance measurement 3.7 Risk analysis and measurement 3.8 Value management 3.9 Change control 3.10 Mobilization
4. General management	4.1 Operations and technical management 4.2 Marketing and sales 4.3 Finance 4.4 Information technology 4.5 Law 4.6 Procurement 4.7 Quality 4.8 Safety 4.9 Industrial relations

E.5 UK National Vocational Qualifications

Another initiative, in the United Kingdom, has been the development of National (and Scottish) Vocational Qualifications (NVQs/SVQs), which assess competence in the work place. For each area of competence, a number of units are specified that describe the functions that candidates should be able to demonstrate an ability to carry out satisfactorily. For each of these units a set of evidence criteria is laid down. This evidence can take the form of actual products, or can come from observation of the candidates carrying out the process. 'Knowledge evidence' can also be collected by questioning the candidate, either orally or by written examination. A set of such standards have been developed by the Occupations Standards Council for Engineering (OSCEng) for project management NVQ/SNQ levels 4 and 5. Figure E.3 provides an overview of the units and elements involved at level 5.

Level 4 can be judged to be the 'entry level' for project management professionals, while it has been suggested that level 5 is at the level that someone taking the Project Executive role (in PRINCE 2 terms) should be able to demonstrate competence.

For each element, a set of *performance criteria* has been drafted, supported by *range statements*, which define the scope of the terms used in the criteria. For example, under 'secure materials, equipment and facilities to implement project', there is a performance criterion a fragment of which is:

'sources of materials, equipment and facilities ... are identified'

In the accompanying range statements, the scope of 'materials', 'equipment' and 'facilities' is defined. For example, 'facilities' includes 'accommodation and related services, servicing and maintenance, telecommunications, transport and technical/administrative support'. For each set of criteria, *evidence requirements* are defined.

In the United Kingdom, the scheme seems to us to represent the most comprehensive and well-thought out.

There are five bodies in the United Kingdom who carry out assessment for this qualification, including the Institute of Management.

The Institute of Management also administer a Certificate and a Diploma in project management, which are based taught courses that complement the NVQ/SVQ programme.

Further details of the scheme can be obtained from:

The Institute of Management
Cottingham Road
Corby
Northants
NN17 1TT
UK

Table E.3 *Project management NVQ/SVQ Level 5 summary***Develop objectives for projects**

- Define clients' aims and initial objectives for the project
- Identify and assess factors affecting the achievement for the project
- Develop and agree objectives for the project

Specify requirements for projects

- Develop means of implementing the project to achieve the objectives
- Produce and agree specification of requirements for a project

Estimate resources and develop programmes for projects

- Specify and estimate resources required for the project
- Develop programme and schedules for the project

Recommend contracting arrangements for projects

- Identify the means of providing the resources to implement the project
- Select and agree a procurement strategy and procedure(s)
- Recommend type(s) and forms(s) of contract for the project

Secure resources to implement projects

- Secure finance for the project
- Ensure the availability of specified personnel to implement the project
- Secure materials, equipment and facilities to implement the project
- Acquire information needed to implement the project

Control risk in implementing projects

- Establish and maintain a culture of risk awareness
- Identify potential risks and evaluate options for their control
- Select options and implement measures for controlling risks
- Monitor risks and review the effectiveness of measures for controlling them

Establish the project organization

- Establish the project team
- Establish and monitor team working methods
- Verify systems for managing project health, safety and welfare

Control implementation of projects

- Establish procedures and responsibilities for the project
- Monitor, co-ordinate and control the project
- Comply with regulatory requirements
- Monitor and control income and expenditure in implementing the programme
- Ensure quality in the implementation of the project
- Control handover of responsibilities for facilities and projects

Evaluate project achievements and secure improvements

- Obtain and evaluate feedback information
- Provide advice and support to solve problems, make improvements and maintain progress
- Promote and protect planned work and those who carry it out

E.6 Information Systems Examination Board

A personal interest needs to be declared here as one of the authors has been involved as an examiner and moderator with this body.

The Information Systems Examination Board (ISEB) is the wing of the British Computer Society (BCS) that sets and administers examinations to assess professional competence in a number of fields related to information technology and information systems development. These fields include Structured Systems Analysis and Design Method (SSADM), IT Service Management, IT Infrastructure Management, and Project Management. Thus, unlike the qualifications we have already looked at, this is targeted specifically at IT staff.

The aim of the Certificate is to provide an entry level qualification for project management. Candidates must have at least four years' experience in management or in information systems and have attended an accredited course in project management where the candidate's coursework will have been assessed. They then have to take a three hour written examination and, if successful, pass an oral examination. There is also an experienced project manager route where the candidate can sit the written and oral examinations without having attended a course.

The syllabus is designed to be delivered via a two-week 80 hour course. A recent development has been the accreditation of some distance learning courses, such as that produced by the Open University. An overview of the syllabus is shown in Table E.4.

In April 1996, the ISEB launched the Diploma in Project Management, which is designed for practitioners who have the Certificate and eight years' minimum experience. Candidates have to submit five written papers of a minimum of 3000 words, which they have to complete in their own time and to make a presentation followed by an oral examination.

It should be noted that the BCS also administer a separate set of professional examinations that include an examination in project management. The pedigree of these examinations is more academic than the ISEB equivalent, which attempt foremost to be industrially relevant. The BCS professional examination syllabus is also more closely focused on *software* project management than the ISEB examination.

Details of these qualifications can be obtained from:

Information Systems Examination Board

1 Sanford Street

Swindon

Wiltshire

SN1 1HJ

UK

Table E.4 ISEB Certificate in Project Management – syllabus outline

<i>Section</i>	<i>Subsection</i>	<i>% of course</i>
Overview	Strategy Planning Organization Risk	7%
Managing plans	Plans Estimating Acceptance of the plan Control Project reviews	15%
Managing people	Skills Organization Human resource management Technical management	15%
Managing other resources	Resources required Additional resources Contracted out resources Resource management	8%
Managing the development and delivery of project products	Identification of products Definition of products Quality specification Product ownership Configuration management Control during production Control after production Product delivery	25%
Managing project documentation	Need for documentation Use of documentation Procedures for updating documentation Documentation roles and responsibilities Types of documentation Types of file	5%
Managing quality	Quality management systems Quality standards Quality plans Quality assurance Quality control	15%
Managing change	Inevitability of change Configuration management Possible sources of change Formal change control Control procedures Implementation of change	10%

Appendix F

Answer pointers

Chapter 1

The order you put these projects is, of course, to a large degree subjective. Here is one example of a possible ordering.

- i. **Building the Channel Tunnel** Almost everybody puts this one first. The huge scale of the task, the relative novelty of the project, all the different specialisms involved and the international nature of the project make it special.
2. **Writing an operating system** This is a prime example of a software development project.
3. **Amending a financial system to deal with dates after 31st December 1999** This project is modifying an existing system rather than creating a new one from scratch. Many software projects have this characteristic and it does not make them any less a software project.
4. **Installing a new version of a word processing package in an organization** Although no software is being produced or modified, many of the stages that are associated with software projects will be involved and the techniques of software project management would be appropriate.
5. **Investigation into the reasons why a user has a problem with a computer system** This will have many of the stages common to software projects, although the precise nature of the end result is uncertain at the outset. It could be that the user needs some simple remedial training. On the other hand, it could turn out to be quite a considerable software modification task.
6. **Getting married** There should be lots of arguments about this one! Some will be reluctant to give a high rating to this because of its personal nature. The degree to which this is 'project-like' will depend very much upon the cultural milieu in which it takes place. Very often it requires a high degree of planning, involves lots of different people and, for most people, is a non-routine operation.
7. **A research project into what makes a good human-computer interface** Compared to some of the projects above, the objectives of the research project are more open-ended and the idea of a specific client for the end product may

1.1 Examples of projects

be less well-defined. Research projects are in some ways special cases and the approach to their planning needs a rather different approach, which is outside the scope of this book.

8. **Producing an edition of a newspaper** In some ways this has all the characteristics of a project. There are lots of different people with lots of different specialisms whose work needs to be coordinated in order to produce an end product under very tight time constraints. What argues against this as a typical project is that it is repeated. After a while, everyone knows what they each need to do and most of the problems that arise are familiar and the procedures to deal with them are well-defined.
9. **A second year programming assignment for a computing student** This is not being done for a customer, although it could be argued that the tutor responsible for setting and assessing the assignment is, in effect, a surrogate client. Not all the stages of a normal project will be gone through.

1.2 Brightmouth College payroll: Stages of a project

1. **Project evaluation** All the costs that would be incurred by the college if it were to carry out its own payroll processing would need to be carefully examined to ensure it would be more cost effective than letting the local authority carry on providing the service.
2. **Planning** The way that the transfer to local processing is to be carried out needs to be carefully planned with the participation of all those concerned. Some detailed planning would need to be deferred until more information was available, for example, which payroll package was to be used.
3. **Requirements analysis** This is finding out what the users need from the system. To a large extent it will often consist of finding out what the current system does, as it may be assumed that in general the new system is to provide the same functions as the old. The users might have additional requirements, however, or there might even be facilities that are no longer needed.
4. **Specification** This involves documenting what the new system is to be able to do.
5. **Design/coding** As an 'off-the-shelf' package is envisaged, these stages will be replaced by a package evaluation and selection activity.
6. **Verification and validation** Tests will need to be carried out to ensure that the selected package will actually do what is required. This task might well involve parallel running of the old and new systems and a comparison of the output from them both to check for any inconsistencies.

7. **Implementation** This would involve such things as installing the software, setting system parameters such as the salary scales, and setting up details of employees.
8. **Maintenance/support** This will include dealing with users' queries, liaising with the package supplier and taking account of new payroll requirements.

Many large organizations that are committed to computer-based information systems have specialists responsible for the maintenance of operating systems. However, as an operating system is primarily concerned with driving the hardware it is argued that it has more in common with what we have described as embedded systems.

1.3 The nature of an operating system

This project is really driven by objectives. If in-house payroll processing turns out not to be cost effective, then the project should not try and implement such a solution. Other ways of meeting the objectives set could be considered: for example, it might be possible to contract out the processing to some organization other than the local authority at a lower cost.

1.4 Brightmouth College payroll: objectives-driven vs. product-driven

The danger here is to think only in terms of software modules. The payroll system will contain both technical and human elements. A breakdown into subsystems

1.5 Brightmouth College payroll: subsystems

will vary tremendously according to your particular viewpoint. Figure F.1 is a diagrammatic representation of just one possible answer.

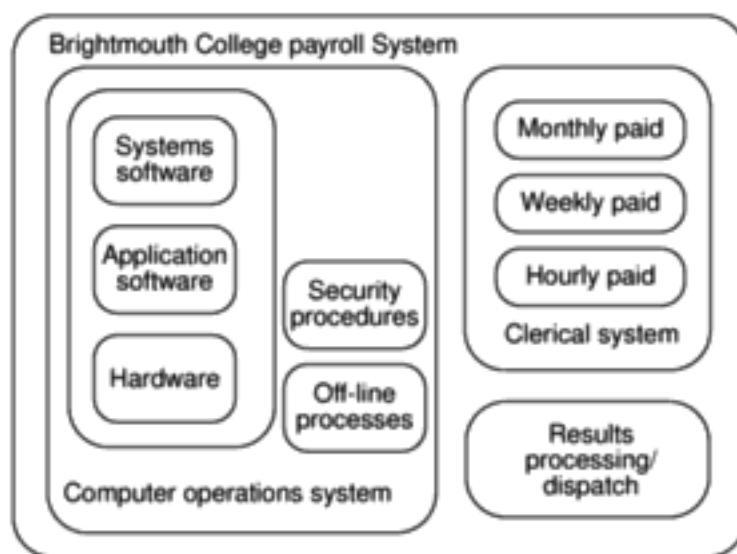


Figure F.1 A systems map of Brightmouth College payroll.

The environment of the system would contain:

- tax authorities, such as the *Inland Revenue and Contributions Agency* in the UK;
- pension funds;
- banks for arrangement of payment by EDI (Electronic Data Interchange);
- trades unions (staff may have subscriptions deducted at source);
- software suppliers;
- hardware suppliers;
- other office equipment suppliers;
- a security firm if some staff are paid in cash;
- external auditors;
- college management;
- site management (who are responsible for physical accommodation);
- staff.

1.6 A day in the life of a project manager**Planning:**

- staffing requirements for the next year.

Representing the section:

- at the group meeting;
- when communicating with the personnel manager about replacement staff;
- when explaining about the delay to users.

Controlling, innovating, directing:

- deciding what needs to be done to make good the progress that will be lost through temporarily losing a member of staff.

Staffing:

- deciding which member of staff is to do what;
- discussion with personnel about the requirement for temporary staff;
- planning staffing for the next year.

Note: the same activity can involve many different roles.

1.7 Brightmouth college payroll: objectives, goals and measures of effectiveness

The original objective might have been formulated as: 'To carry out payroll processing at less cost while maintaining the current scope and quality of services'.

In order to achieve this, sub-objectives or goals will usually have been identified, for example:

- to transfer payroll processing to the college by 1st April;
- to implement in the new system those facilities that exist in the current system less those identified in the initial report as not being required;
- to carry out the implementation of the payroll processing capability within the financial constraints identified in the initial report.

It should be noted that the objectives listed above do not explicitly mention such things as putting into place ongoing arrangements to deal with hardware and software maintenance, security arrangements and so on. By discussing and trying to agree objectives with the various people involved the true requirements of the project can be clarified.

Measures of effectiveness for the sub-objectives listed above might include the following:

- **Date of implementation** Was the new system being used operationally by the agreed date?
- **Facilities** In parallel runs, were all the outputs produced by the old system and still required also produced by the new system?
- **Costs** How did the actual costs incurred compare with the budgeted costs?

1.8 Brightmouth college payroll: stakeholders

Most of the external entities identified in Exercise 1.5 would also be stakeholders in the project.

Major stakeholders would include:

- the finance department;
- the personnel department, who would need to supply most of the employee details needed;
- heads of departments, who would need to submit details of hours worked for part-time staff;
- staff, who would naturally be concerned that they are paid correctly;
- site management: the new arrangements may mean that the office layout has to be rearranged physically;
- software and hardware vendors.

One group of stakeholders that might not be readily identified at first is the local government authority and its staff. It might seem strange to list the people who used to do the job, but who are no longer required. The project manager's job will be made a lot easier if their cooperation and help can be obtained. The project manager would do well to sound out tactfully how the local authority staff feel about losing this work. It could be that they are pleased to be shot of the workload and hassle involved! Arrangements that take into account existing local authority staff might be possible. For example, if the college needs to recruit new staff to deal with payroll, it might smooth things to give the job to a member of the local authority staff who already deals with this area.

Chapter 2

The main stakeholders who needs to be considered are the IOE customers. It will be worth consulting some representative customers about the layout of the new monthly statement for example.

2.1 External stakeholders in IOE accounts system

Figure F.2 illustrates the product flow diagram for invitation to tender for Brightmouth College payroll.

2.2 Invitation to tender PDF

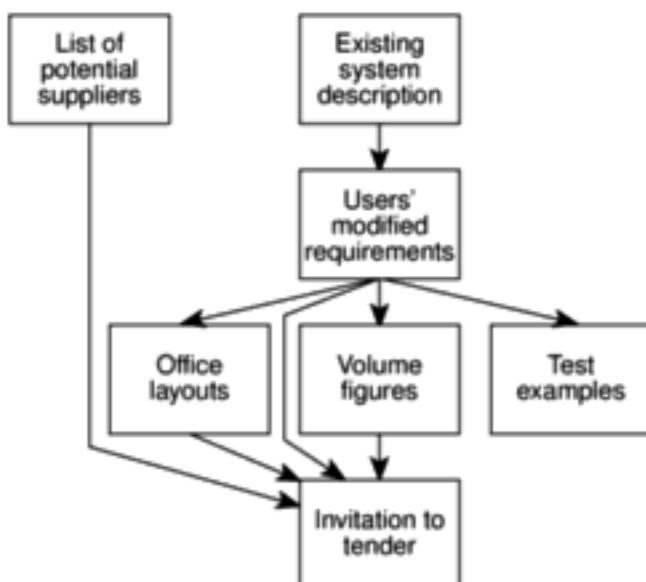


Figure F.2 Product flow diagram for 'Invitation to tender'.

Figure F.3 illustrates the activity network for invitation to tender for the Brightmouth College payroll.

2.3 Invitation to tender activity network

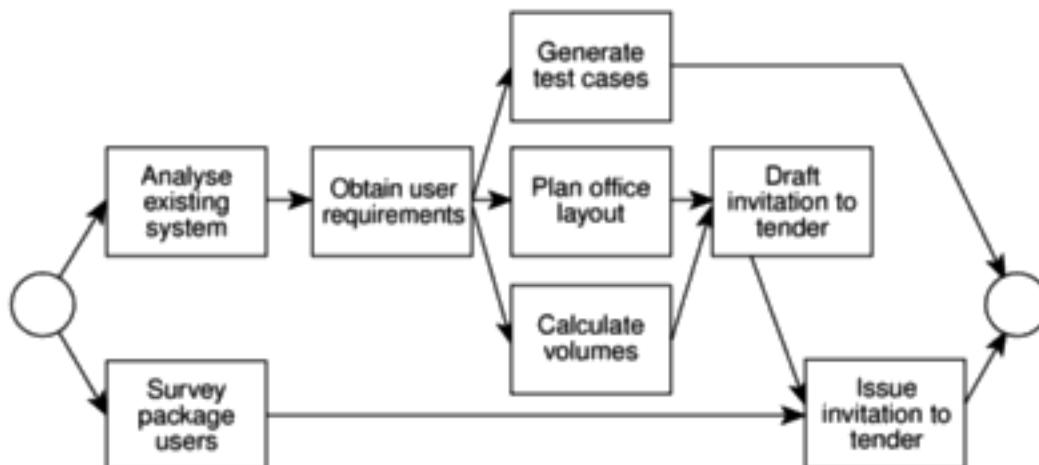


Figure F.3 Brightmouth College payroll project activity network fragment.

2.4 Including a checkpoint

Figure F.4 illustrates the inclusion of a checkpoint in Amanda's activity network.

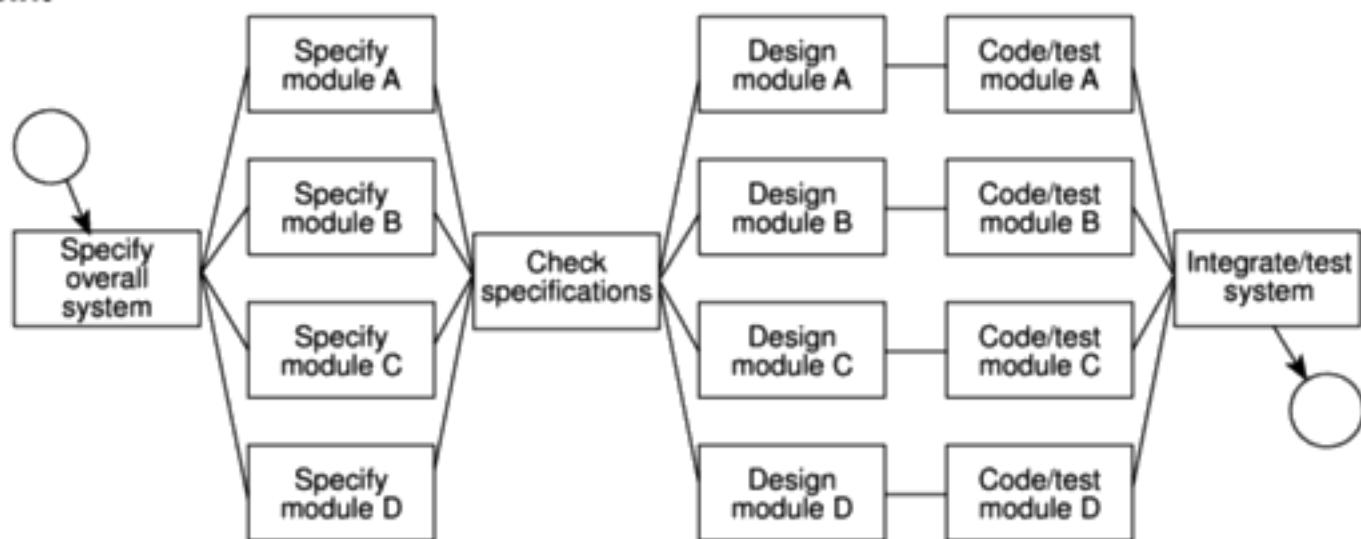


Figure F.4 Fragment of the IOE maintenance group accounts activity network.

2.5 Quality checks on user requirements

The users will need at least to read and approve the system specification. This might be rather late to make major changes, so user approval of earlier documents such as interview notes would be helpful.

Chapter 3

Table F.1 lists costs and benefits for the proposed Brightmouth HE College payroll system. It is not comprehensive but illustrates some of the types of items that you should have listed.

3.1 Costs and benefits for the Brightmouth College payroll system

Table F.1 *Costs and benefits for the Brightmouth College payroll system*

Category	Cost/benefit
Development costs	software purchase – software cost plus selection and purchasing cost project team employment costs
Setup costs	training include costs of trainers and operational staff time lost while training staff recruitment computer hardware and other equipment which might have a residual value at end of projected life accommodation – any new/refurbished accommodation and furniture required to house new system. initial systems supplies – purchase of stationery, disks and other consumables
Operational costs	operations staff – full employment costs stationery – purchase and storage* maintenance and standby – contract or estimation of occurrence costs accommodation including heating, power, insurance etc.*
Direct benefits	saving on local authority fees later payment – increase interest income through paying salaries later in the month
Indirect benefits	improved accuracy – assumes that direct costs of correcting current errors that should not occur with a computerized system are known (for example, takes one person one day per week) Note: benefit should measure what can be done with that additional time
Intangible benefits	improved management information – this should lead to improved decision making but it is very difficult to quantify the potential benefits

* These items, and some other elements, might show corresponding savings or costs through no longer being required. For example, although new office furniture might be required for the new system, the existing furniture might be redeployed or sold

3.2 Ranking project cash flows

Obviously you will have your own views about which have the best and worst cash flows. You should, however, have considered the following points: project 2 requires a very large investment compared to its gain – in fact we could obtain £100,000 by undertaking both projects 1 and 3 for a lower cost than project 2. Both projects 1 and 4 produce the bulk of their incomes relatively late in their lives compared with project 3, which produces a steady income over its life.

3.3 Calculating payback periods

The payback periods for each of the projects will occur during the year indicated: project 1 year 5, project 2 year 5, project 3 year 4 and project 4 year 4 (end).

We would therefore favour project 3 or 4 over the other two. Note that, in reality, with relatively short-term projects such as these we would produce a monthly (or at least quarterly) cash flow forecast and it is therefore likely that project 3 would be seen more clearly to have a shorter payback period than project 4.

3.4 Calculating the return on investment

The return on investments for each of the projects is: project 1: 10%, project 2: 2%, project 3: 10% and project 4: 12.5%. Project 4 therefore stands out as being the most beneficial as it earns the highest return.

3.5 Calculating the net present value

The net present value for each of the projects is calculated as in Table F.2. On the basis of net present value, project 4 clearly provides the greatest return and project 2 is clearly not worth considering.

Table F.2 *Calculating the net present value of projects 2, 3 and 4*

Year	Discount factor	Discounted cash flow (£)		
		Project 2	Project 3	Project 4
0	1.0000	-1,000,000	-100,000	-120,000
1	0.9091	181,820	27,273	27,273
2	0.8264	165,280	24,792	24,792
3	0.7513	150,260	22,539	22,539
4	0.6830	136,600	20,490	20,490
5	0.6209	186,270	18,627	46,568
NPV		-179,770	13,721	21,662

Table F.3 illustrates the effect of varying discount rates on the NPV. In each case the 'best' project is indicated in bold. In this somewhat artificial example, which project is best is very sensitive to the chosen discount rate. In such a case we must either have a very strong reason to use a particular discount rate or take other criteria into account when choosing among the projects.

3.6 Calculating the effect of discount rates on NPV

Table F.3 *The effect on net present value of varying the discount rate*

Year	Cash flow values (£)		
	Project A	Project B	Project C
0	-8,000	-8,000	-10,000
1	4,000	1,000	2,000
2	4,000	2,000	2,000
3	2,000	4,000	6,000
4	1,000	3,000	2,000
5	500	9,000	2,000
6	500	-6,000	2,000
Net Profit	£ 4,000	£ 5,000	£ 6,000
NPV @ 8%	£ 2,111	£ 2,365	£ 2,421
NPV @ 10%	£ 1,720	£ 1,818	£ 1,716
NPV @ 12%	£ 1,356	£ 1,308	£ 1,070

Expected sales of £500,000 per year over four years would generate an expected net income of £1.2m (after allowing for annual costs of £200,000) which, by almost any criteria, would provide a good return on an investment of £750,000. However, if sales are low, and there is a 30% chance of this happening, the company will lose money – it is unlikely that any company would wish to take such a risk knowingly.

3.7 Project evaluation using cost–benefit analysis

This example illustrates one of the basic objections to using this approach for one-off decisions. Were we to repeat the project a large number of times we would expect, *on average*, an income of £500,000 per annum. However, the company is developing this package only once – they can't keep trying in the hope of, on the average, generating a respectable income. Indeed, a severe loss on this project could mean it is the last project they are able to undertake.

Chapter 4

4.1 Classification of systems

- (a) A payroll system is a data-oriented or information system that is application specific.
- (b) The bottling plant system is a process control or industrial system which contains embedded software.
- (c) This looks like an information system that will make heavy use of computer graphics. The plant itself might use control software which might be safety-critical but this is not the subject of the project under consideration.
- (d) Project management software tools are often categorized as general packages. There would be a considerable information systems element to them.
- (e) This could use an information retrieval package that is a general software package. It is also a strong candidate for a knowledge-based system.

4.2 Identification of risks

The user staff could, arguably, be regarded as a project resource. The writers' view is that it is useful to add a fourth category of risks – those belonging to the *environment* in which the system is to be implemented.

Among the risks that might be identified at Brightmouth College are:

- conflict of views between the finance and personnel departments;
- lack of staff acceptance for the system, especially among personnel staff;
- lack of cooperation by the local authority that used to carry out payroll work;
- lack of experience with running payroll at the college;
- lack of administrative computing expertise at the college;
- possible inadequacy of the chosen hardware;
- changes to the payroll requirements.

4.3 Selection of project approaches

- (a) This would appear to be a knowledge-based system that is also safety-critical. Techniques associated with knowledge based systems could be used for constructing the system. Testing would need to be very carefully conducted. A lengthy parallel run where the system is used to shadow the human decisions made in real cases and the results compared could be

considered. Another approach would be to develop two or more systems in parallel so that the advice offered could be cross-checked.

- (b) This is an information system that will be on a relatively large scale. An SSADM approach would be justified. When student loans were first introduced there was no existing system and so there might have been some scope for a prototype.
- (c) This is an embedded system that is highly safety-critical. Measures that might be adopted to ensure the reliability of the system include:
 - use of mathematics-based specification languages to avoid ambiguity;
 - developing parallel versions of the same software so that they can be cross-checked;
 - statistical control of software testing to allow for the estimation of the reliability of the software.

The review might find that the benefits forecast in the original feasibility study report have not been achieved. ‘Corrections’ to the existing system can allow those benefits and other ones to be realized. This would lead to a proposal for a new project to modify the installed option

4.4 Feedback between project review and feasibility study

- (a) A prototype could be useful as part of the feasibility study. A mock-up of an executive information system loaded with current management information could be set up manually and then be tried out by the managers to see how easy and useful they found it.
- (b) A prototype could be used to assist in the design of the user dialogues. SSADM allows for prototypes for this purpose as part of its requirement specification module.
- (c) A prototype of the most response critical transactions could be made at the physical design stage to see whether Microsoft Access could produce software that gave a satisfactory performance.

4.5 Stages of a project where a prototype can be appropriate

Chapter 5

5.1 Calculating productivity rates and using productivity rates to project effort

Tables F.4 and F.5 illustrate productivity rates and estimated project effort.

Table F.4 *Productivity rates*

<i>Project</i>	<i>Work-months</i>	<i>SLOC</i>	<i>Productivity</i> (<i>SLOC/month</i>)
a	16.7	6050	362
b	22.6	8363	370
c	32.2	13334	414
d	3.9	5942	1524
e	17.3	3315	192
f	67.7	38988	576
g	10.1	38614	3823
h	19.3	12762	661
i	59.5	26500	445
Overall	249.3	153868	617

Table F.5 *Estimated effort*

<i>Project</i>	<i>Estimated work-months</i>	<i>Actual</i>	<i>Difference</i>
a	$6050/617 = 9.80$	16.7	6.90
d	$5942/617 = 9.63$	3.9	-5.73

There would be an under-estimate of 6.9 work-months for project *a* and an over-estimate of 5.7 for project *d*.

5.2 Course staff costs program – activities required

A list of activities might include:

- obtain user requirements;
- analyse the structure of the data already held;
- design report and write user proposal;
- write test plan;
- write technical specification;
- design software;
- write software;
- test software;

- write operating instruction;
- carry out acceptance testing.

The most difficult tasks to estimate are often those that are most sensitive to the size and the complexity of the software to be produced, in this case the design, writing and testing of the software. Writing the technical specification can also be difficult because of this, but estimating problems tend to be concealed here as deadlines can be met by omitting detail that can be added latter when deficiencies are found.

The duration of activities that are to be carried out by users may also present problems, as this might depend upon their sense of priorities.

The most obvious effort driver would seem to be the number of words required. Difficulty factors might include:

- **availability of material**, for example, in the library;
- **familiarity** of the student with the topic;
- **breadth/depth** required, that is, a broad survey of a wide field or an in-depth study of a narrow area;
- **technical difficulty**, that is, some topics are easier to explain than others.

5.3 Effort drivers for a student assignment

It could be argued that time available is the constraint. The student just does what can be done in the time available (see 'design to cost').

The Euclidean distance between Project B and the target case

5.4 Calculating Euclidean distance

$$= \sqrt{(7 - 5)^2 + (15 - 10)^2}$$

$$= \sqrt{2^2 + 5^2}$$

$$= 5.39$$

Project A is therefore a closer analogy.

5.5 Albrecht function points

The function types are as follows.

External input types	none
External output types	the report, 1
Logical internal file types	the accounting feeder file, 1
External interface file types	payroll file, staff file (timetabling,) courses file (timetabling,) accounting feeder file, 4
External inquiry types	none

Because the accounting feeder file is outgoing, it is counted once as a logical internal file type and once as an external interface file type.

External enquiry types	none
External output types	$1 \times 7 = 7$
Logical internal file types	$1 \times 10 = 10$
External interface types	$4 \times 7 = 28$
External inquiry types	none

Total	45
-------	----

5.6 Calculation of SLOC from Albrecht function points

Estimated lines of Cobol = $45 \times 91 = 4095$

5.7 Mark II function points

The function types are:

Input data types	6
Entities accessed	1
Output data types	1

Unadjusted function points = $(0.58 \times 6) + (1.66 \times 1) + (0.26 \times 1) = 5.4$.

5.8 SLOC estimate for customer insertion program

Figure F.5 gives an outline program structure. The numbers in circles are our estimates of the lines of Cobol code needed to implement each subprocess in the program. They should add up to 95 SLOC. Note that these do not include data declarations.

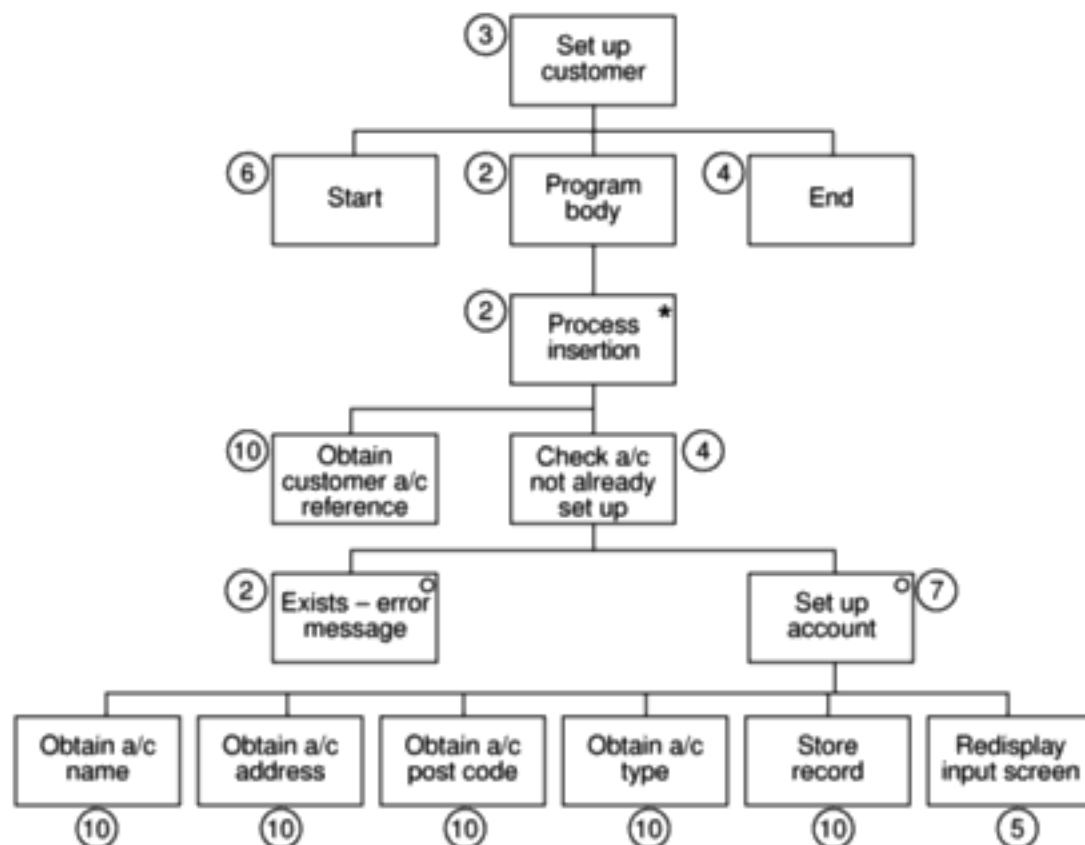


Figure F.5 Outline program structure for 'set up customer' transaction.

Table F.6 shows a comparison of the actual work-months from Table 5.1 and the COCOMO estimates. It illustrates the need for the calibration of COCOMO to suit local conditions.

5.9 COCOMO estimates

Table F.6 Comparison of COCOMO estimates and actual effort

SLOC	Actual (work-months)	COCOMO estimates	Difference (work months)	Difference (%)
6050	16.7	15.9	-0.8	-4.9
8363	22.6	22.3	-0.3	-1.2
13334	32.2	36.4	4.2	13.1
5942	3.9	15.6	11.7	299.7
3315	17.3	8.4	-8.9	-51.2
38988	67.7	112.4	44.7	66.0
38614	10.1	111.2	101.1	1001.5
12762	19.3	34.8	15.5	80.2
26500	59.5	74.9	15.4	25.9

5.10 COCOMO – calculating the development multiplier

Table F.7 shows development effort multipliers (*dem*) for the IOE project.

Table F.7 *Calculating the development multiplier*

Factor	Rating	Multiplier
ACAP	very high	0.71
AEXP	low	1.13
PCAP	high	0.80
VEXP	high	0.90
LEXP	low	1.07

$$dem = 0.71 \times 1.13 \times 0.80 \times 0.90 \times 1.07 = 0.62$$

$$\text{final estimate} = 4 \text{ person-months} \times 0.62 = 2.48 \text{ staff months}$$

5.11 COCOMO II Maximum size of sf

The maximum possible value for *sf* is:

$$1.01 + 0.01 \times (5 + 5 + 5 + 5 + 5) = 1.26$$

5.12 COCOMO II calculating a scale factor

Table F.8 illustrates the exponent drivers and ratings for this project.

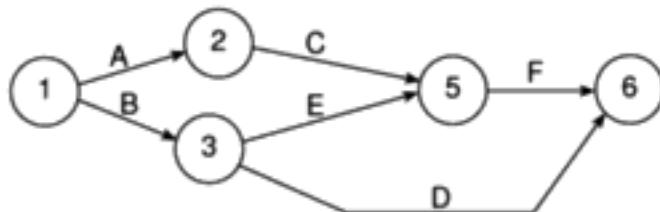
Table F.8 *Exponent drivers and ratings for the project*

Exponent driver	rating
Precedentedness	3
Development flexibility	0
Architecture/risk resolution	4
Team cohesion	1
Process maturity	4
Total	12

$$\text{Therefore, } sf = 1.01 + 0.01 \times (3 + 0 + 4 + 1 + 4) = 1.13$$

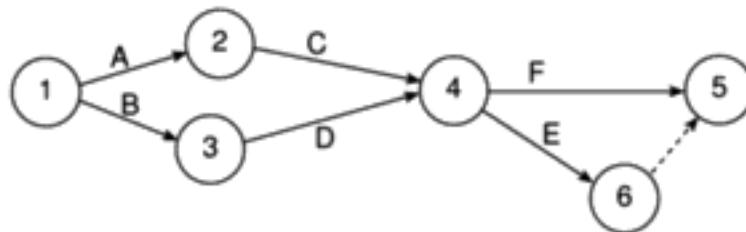
Chapter 6

- (a) Activity D dangles, giving the project two 'end events'. This network should be drawn as below. To aid comparison with the original, the nodes have not been renumbered, although we would normally do so.

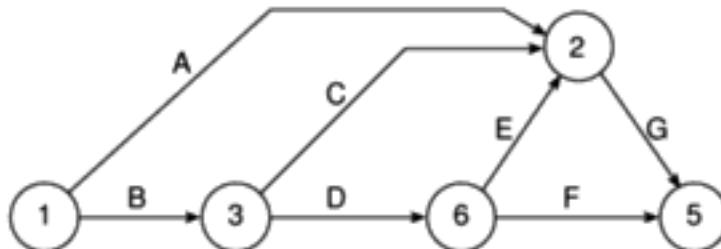


6.1 Errors drawing activity networks

- (b) Once again, this network has two end nodes, but in this case the solution is slightly different since we should introduce a dummy activity if we are to follow the standard CPM conventions.



- (c) Either this one has a dangle (although, because of the way it is drawn, it is less obvious) or activity E has its arrow pointing in the wrong direction. We need a bit more information before we can redraw this one correctly.
- (d) Strictly speaking, there is nothing wrong with this one – it is just badly drawn and the nodes are not numbered according to the standard conventions. It should be redrawn as in the following example.



In this diagram the nodes have retained their original numbers (to aid identification) although they should of course be renumbered sequentially from left to right.

- (e) This one contains a loop – F cannot start before G has finished, G cannot start before E has finished and E cannot start before F has finished. One of the arrows is wrong! It is probably activity F that is wrong but we cannot be sure without further information.

6.2 Drawing Brigette's activity network as a CPM network

Brigette's payroll CPM network should look like the diagram below. If your diagram is not exactly the same as this check that it is logically the same.

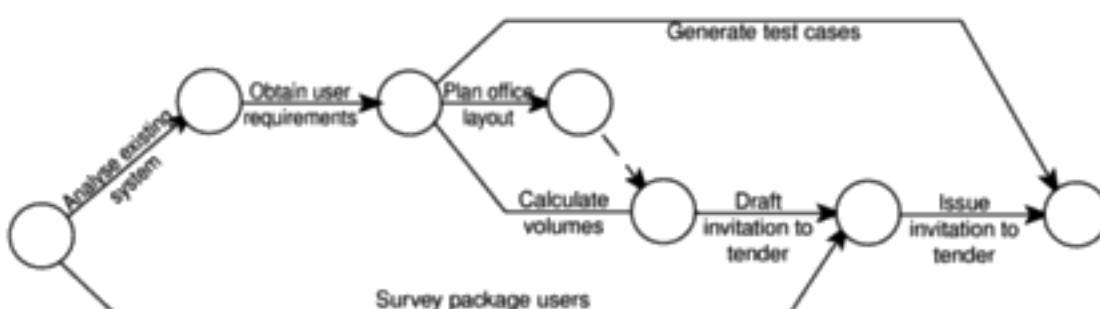


Figure F.6 *Brigette's CPM network.*

6.3 Drawing a CPM network

A solution is given in Figure 6.3. If your solution is not exactly the same as this do not worry. Just check that it is *logically* the same and that it follows the CPM conventions of layout and node numbering etc.

6.4 Calculating activity floats

Free float and interfering float for each of the activities are shown in Table F.9 below. Note that activity A has no free float since any delay in its completion will delay the start of activity C. Activity C, however, has a 2-week free float so long as activity A keeps to time. Float must be regularly monitored as a project progresses since a delay in any activity beyond its free float allowance will eat into the float of subsequent activities.

Table F.9 *Activity floats*

Activity	Total float	Free float	Interfering float
A	2	0	2
B	3	0	3
C	2	2	0
D	3	3	0
E	3	3	0
F	0	0	0
G	0	0	0
H	2	2	0

6.5 Shortening a project duration

Shortening activity F to 8 weeks will bring the project completion date forward to week 11 – that is, it will save 2 weeks on the duration of the project. However,

there are now two critical paths, 1–5–6 and 1–2–4–6, so that reducing the duration of activity F any further will not shorten the project duration any further. If we wish to complete the project earlier than week 11 we must save time on path 1–5–6 and path 1–2–4–6.

6.6 The precedence network

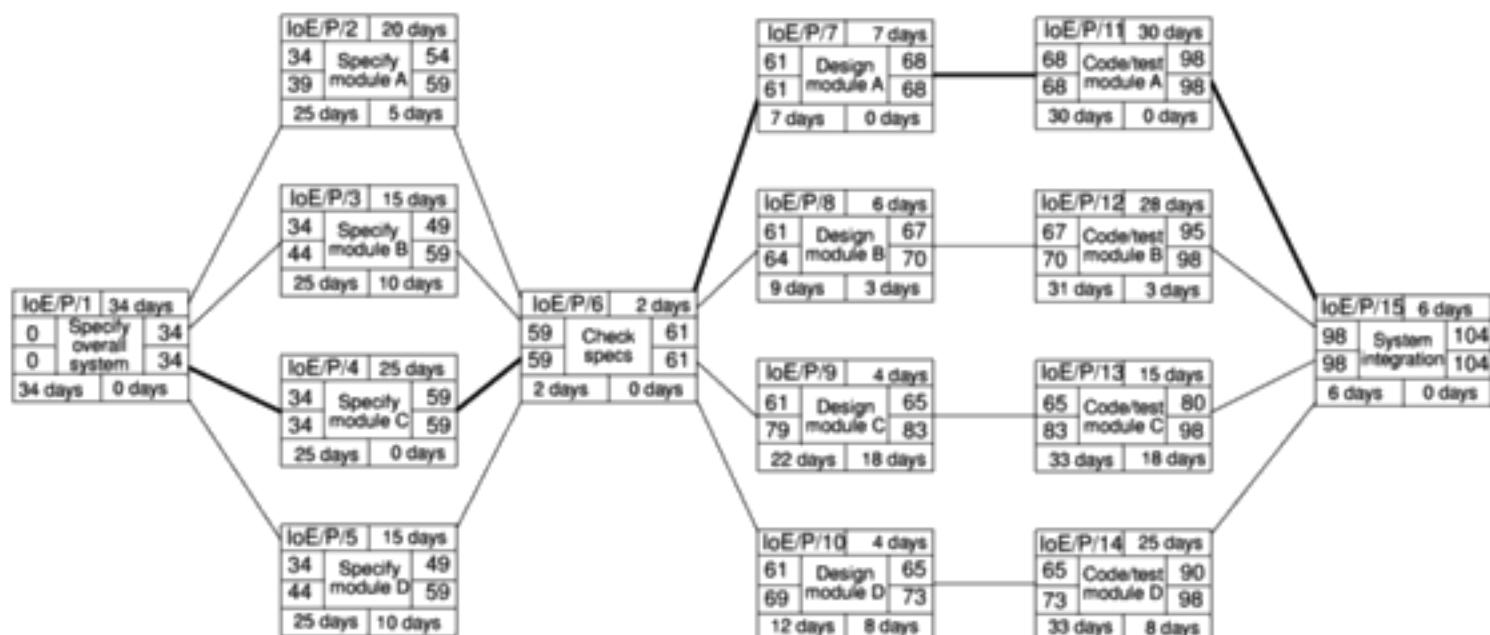


Figure F.7 Amanda's precedence network.

Chapter 7

7.1 Brigitte's hazard questionnaire

Among the factors that she might consider are the following.

Application factors

- Will the package need to interface with other (existing) systems?
- Are there likely to be large differences between alternative packages?

Staff factors

- Do the staff at the college have experience in evaluating or procuring packages of this type?
- Do the Brightmouth College staff have experience of using similar packages?

Project methods

- Can we use standard methods for this project?
- Does the college have established procedures for this type of project?

Hardware factors

- Will the project involve the purchase of new hardware?
- Will we be able to test candidate packages on the same hardware configuration as we will be operating?

Changeover factors

- Can we run a pilot system before complete changeover?
- Will the master files be convertible from the existing system?

Supplier factors

- Have we experience of purchasing software/hardware from the likely suppliers?
- How well established are the suppliers we are considering?

Environment factors

- Are there any plans for reorganization within the college that could affect the system?
- Are there likely to be any changes in government legislation that could affect the project?

7.2 Amanda's risk prioritization

The risks that you add will clearly depend upon the answers you gave to the previous exercise and you might feel that your assessment of likelihood and impact is somewhat arbitrary. It is best to work in a small group when trying to

estimate these figures – where you disagree with your colleagues over a figure then discuss why this is so. For this exercise, the exact numbers that you decide on are not as important as recognizing why you might not agree with each other.

Categorizing risk exposures is discussed in the paragraphs following the exercise and, although you will have more risks than discussed there, you should look for sensible cut-off points along the lines discussed.

The following are illustrative of the actions that Amanda might consider.

- Changes to requirements specification during coding. This risk could be reduced by ensuring the original specification agreed at a senior level and adopting a high change threshold.
- Specification takes longer than expected. Review time estimates or break the activity down into smaller components and estimate each of them. Draw up contingency plans for shortening critical activities later in the project.
- Staff sickness affecting critical activities. Check availability of suitable agency analysts and programmers.
- Staff sickness affecting non-critical activities. Draw up rota of stand-by staff who might be recruited from other projects.
- Module coding takes longer than expected. Scrutinize estimating procedures and compare estimates with similar past projects.
- Module testing demonstrates errors or deficiencies in design. Use more stringent methods to validate design – formal methods or structured walkthroughs could be appropriate.

7.3 Amanda's risk reductions strategies

Table F.10 shows the activity duration estimates from Table 7.3 along with the calculated expected durations, t_e .

7.4 Calculating expected activity durations

Table F.10 *Calculating expected activity durations*

Activity	Activity durations (weeks)			
	Optimistic (a)	Most likely (m)	Pessimistic (b)	Expected (t_e)
A	5	6	8	6.17
B	3	4	5	4.00
C	2	3	3	2.83
D	3.5	4	5	4.08
E	1	3	4	2.83
F	8	10	15	10.50
G	2	3	4	3.00
H	2	2	2.5	2.08

7.5 The forward pass to calculate expected completion date

The expected duration and the expected dates for the other project events are shown in Figure 7.3. An expected duration of 13.5 weeks means that we expect the project to be completed half way through week 14, although since this is only an expected value it could finish earlier or later.

7.6 Calculating standard deviations

The correct values are shown in Figure 7.4. Brief calculations for events 4 and 6 are given here.

Event 4: Path A + C has a standard deviation of $\sqrt{(0.50^2 + 0.17^2)} = 0.53$

Path B + D has a standard deviation of $\sqrt{(0.33^2 + 0.25^2)} = 0.41$

Node 4 therefore has a standard deviation of 0.53.

Event 6: Path 4 + H has a standard deviation of $\sqrt{(0.53^2 + 0.08^2)} = 0.54$

Path 5 + G has a standard deviation of $\sqrt{(1.17^2 + 0.33^2)} = 1.22$

Node 6 therefore has a standard deviation of 1.22.

7.7 Calculating z values

The z value for event 5 is $\frac{10 - 10.5}{1.17} = -0.43$, for event 6 it is $\frac{15 - 13.5}{1.22} = 1.23$

7.8 Obtaining probabilities

Event 4: The z value is 1.89 which equates to a probability of approximately 3%. There is therefore only a 3% chance that we will not achieve this event by the target date of the end of week 10.

Event 5: The z value is -0.43 which equates to a probability of approximately 67%. There is therefore a 68% chance that we will not achieve this event by the target date of the end of week 10.

To calculate the probability of completing the project by week 14 we need to calculate a new z value for event 6 using a target date of 14. This new z value is

$$z = \frac{14 - 13.5}{1.22} = 0.41$$

This equates to a probability of approximately 35%. This is the probability of not meeting the target date. The probability of meeting the target date is therefore 65% (100% - 35%).

Chapter 8

Smoothing analyst–designer demand for stage 4 is reasonably easy. The design of module D could be scheduled after the design of module C. Stage 2 is more problematic as scheduling the specification of module D to start after the completion of B would delay the project. Amanda might consider doing this if whoever is specifying module A could also be allocated to module D for the last six days – although she may well decide that drafting an extra person in to a specification activity is unsatisfactory.

If the activities are scheduled at the earliest dates, then the plan still calls for four analyst–designers as shown in Figure F.8. By delaying the start of some activities, however, Amanda is able to ensure that using three analyst–designers are sufficient except for a single day. This is shown in Figure F.9.

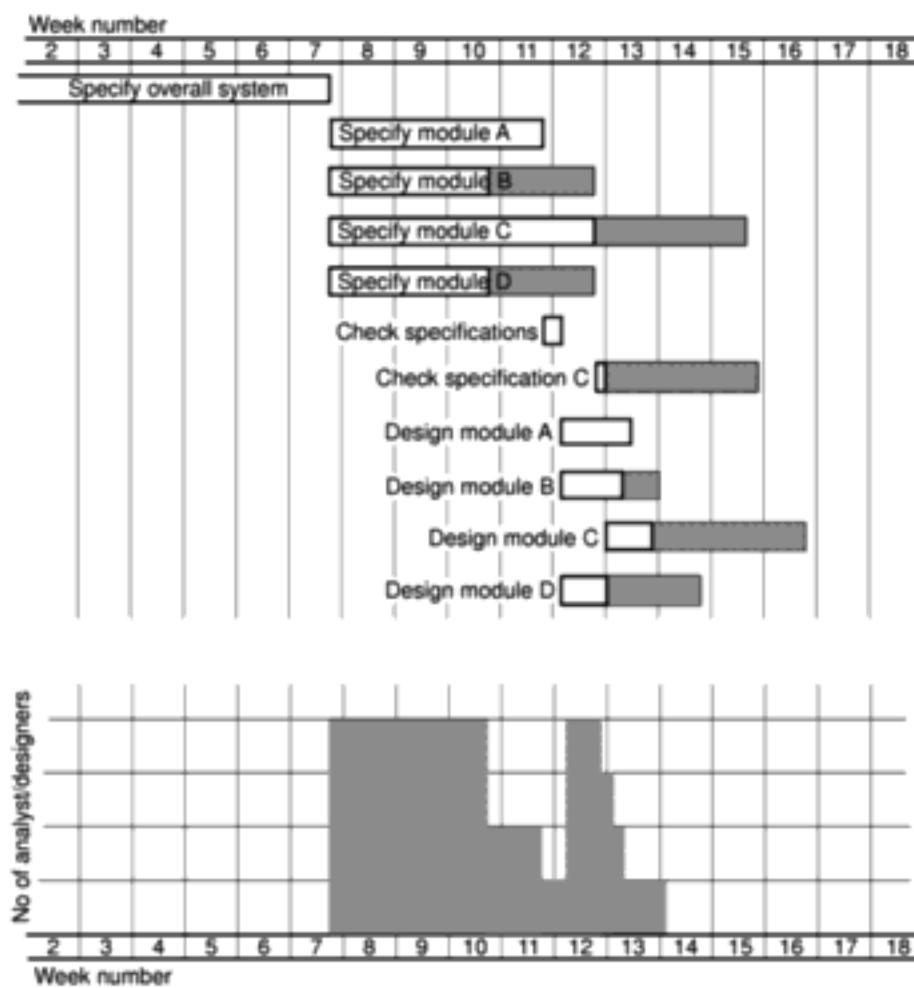


Figure F.8 Amanda's revised bar chart and resource histogram.

Note that if the specification of module C were to be delayed for a further day, the project could be completed with only three analyst–designers, although its completion day would, of course, be delayed.

8.1 Smoothing resource demand

8.2 Drawing a revised resource histogram

When activities are scheduled at their earliest start dates, the shaded area of each bar represents the activity's total float.

Once an activity is scheduled to start later than its earliest date, part of its total float will be 'used up' by that delay. The amount of total float that has been consumed in this way is indicated by the left hand portion of an activity's shaded bar.

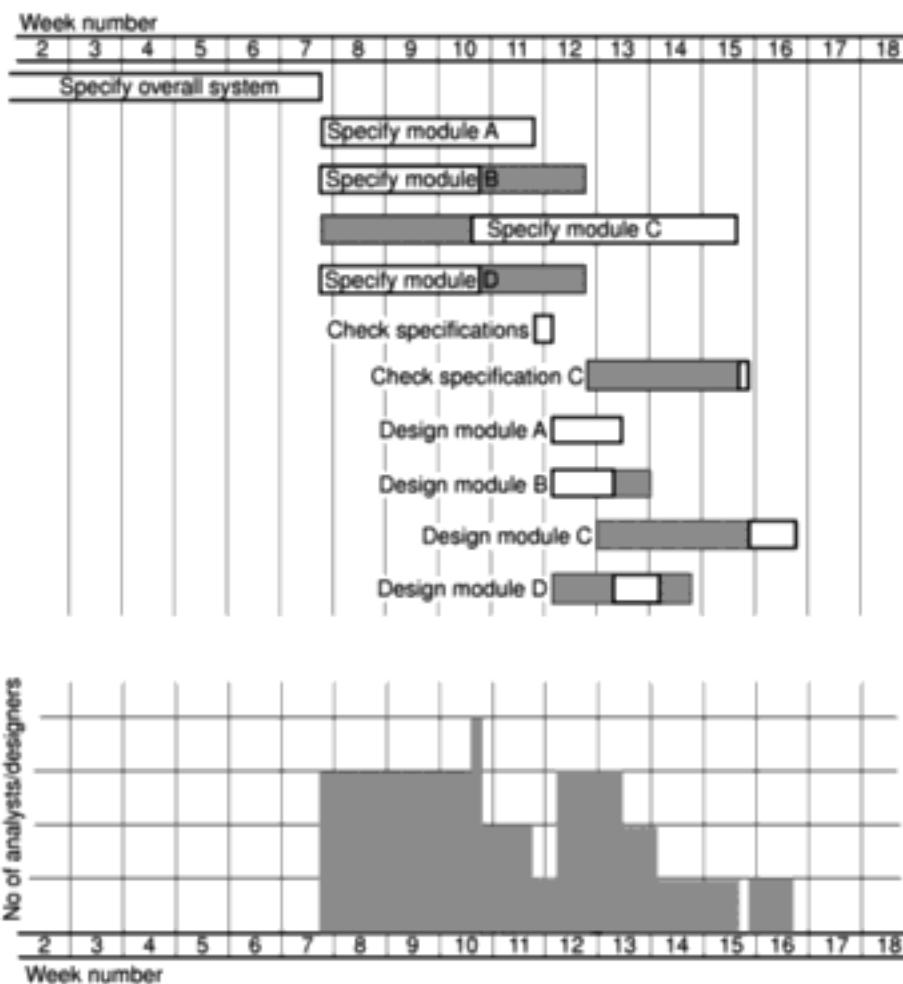


Figure F.9 The effect of delaying some activity starts.

8.3 Identifying critical activities

The critical path is now as shown in Figure F.10. Note the lag of 15 days against activity IoE/P/4, ensuring that its start is delayed until an analyst/designer is expected to be available.

However, the availability of an analyst/designer for IoE/P/4 is dependent upon IoE/P/3 or IoE/P/5 being completed on time – these two activities are therefore also now critical in the sense that a delay in both of them would delay IoE/P/4, which is on the normal critical path. These two activities, although not on the critical path, are, in that sense, critical.

8.4 Assigning staff to activities

Belinda must specify module B as she will then be available in time to start the specification of module C. This leaves Daisy for the specification and design of module A. Belinda cannot do the design of module B as she will still be working on the module C specification when this needs to be done (6 days between days 56 and 66). This will have to be left to Tom, as he should be free on day 60.

Can you think of any other way in which she might have allocated the three team members to these activities?

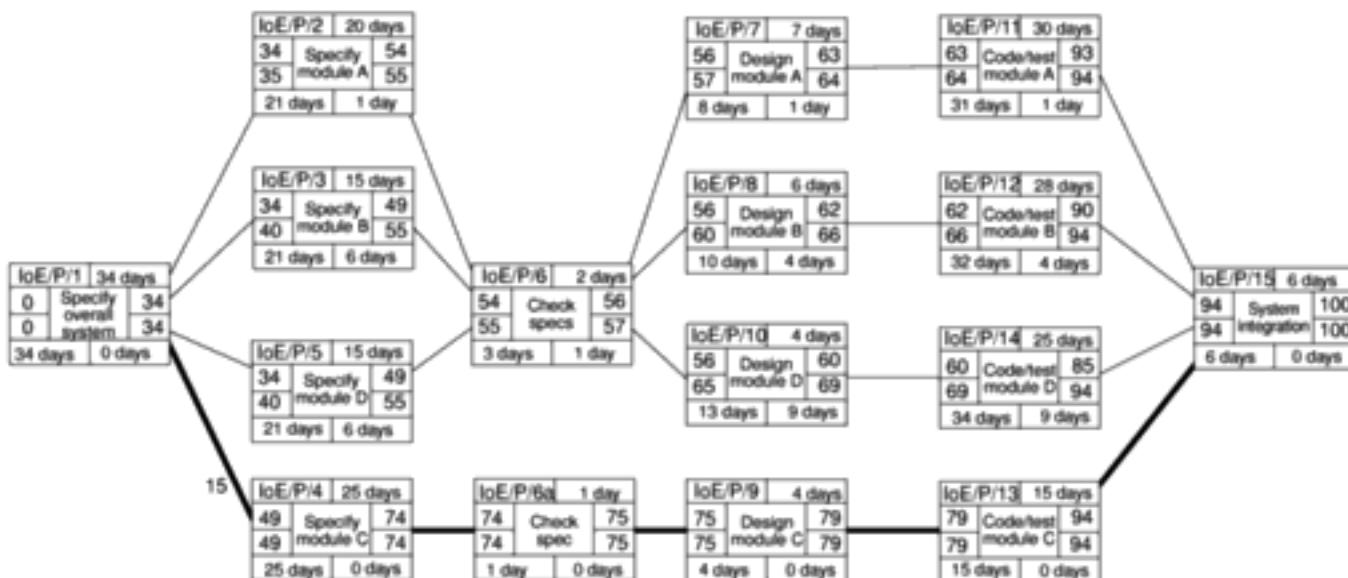


Figure F.10 The critical activities after delaying the start of module C.

The easiest way to calculate the total cost is to set up a table similar to Table F.11.

8.5 Calculating project costs

Table F.11 Calculating the cost of Amanda's project

Analyst	Daily cost (£)	Days Required	Cost (£)
Amanda	300	110*	33,000
Belinda	250	50	12,500
Tom	175	25	4,375
Daisy	225	27	6,075
Gavin	150	30	4,500
Purdy	150	28	4,200
Justin	150	15	2,250
Spencer	150	25	3,750
Daily oncost	200	100	20,000
Total			90,650

* This includes 10 days for pre-project planning and post project review.

Calculating the distribution of costs over the life of the project is best done as a per week or per month figure rather than as daily costs. The expenditure per week for Amanda's project is shown as a chart in Figure 8.9.

Chapter 9

9.1 Lines of code as a partial task completion indicator

There are many reasons why the proportion of lines coded is not a good indicator of completeness. In particular, you should have considered the following:

- the estimated total number of lines of code might be inaccurate;
- the lines of code so far written might have been easier, or harder, to write than those to follow;
- a program is not generally considered complete until it has been tested – when 100% of its lines of code have been written a program will still be incomplete until tested.

With more knowledge of what has been done and what is left to complete it might be possible to make a reasonable estimate of completeness. Breaking the development task into smaller sub-tasks such as software design, coding and unit testing might be of some assistance here.

9.2 Revising the timeline chart

At the end of week 8, the scheduled completion dates for drafting and issuing the tender need to be revised – note both need to be changed since they are both on the critical path (Figure F.11).

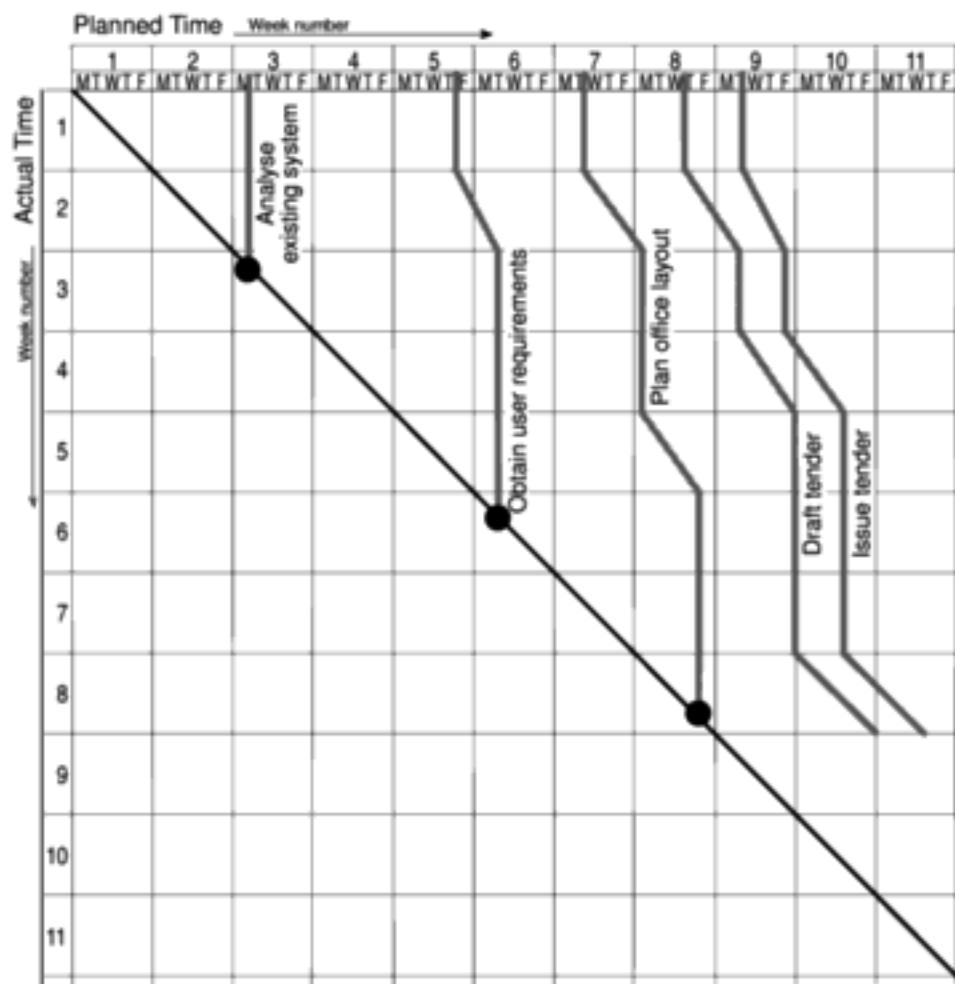


Figure F.11 The revised timeline chart.

Subsequently, Brigette needs to show only the completion of each of these two remaining activities on the timeline chart – the project being completed by the Thursday of week 11 (Figure F.12).

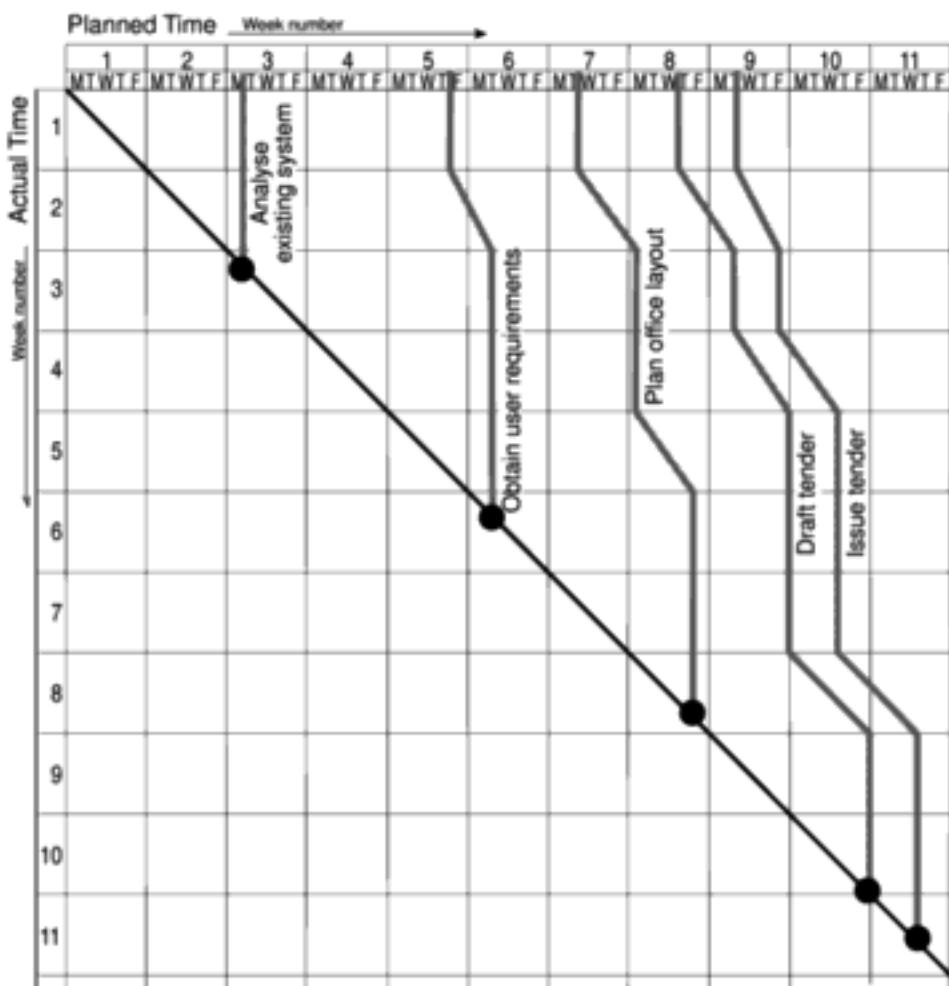


Figure F.12 The completed timeline chart.

It should be apparent from Figure 9.12 that the initial activity, *specify overall system*, has slipped by one day. It may not be quite so obvious from Figure 9.12 alone what else has happened to her project – inspection of Figure 9.12 and Table 9.2 should, however, make it possible to see that *specify module D* has taken 2 days longer than forecast and *specify module B* has taken 5 days longer. Thus, the project has earned 34 workdays by day 35, 49 workdays by day 52 and 64 workdays by day 55.

From Figure 9.12 it is not possible to deduce the underlying causes of the slippage or to forecast the consequences for the project. The use of earned value analysis for forecasting is described later in Section 9.6.

9.3 Amanda's earned value analysis

9.4 The effects of specification changes

Among the items most likely to be affected by the change are test data, expected results and the user handbooks.

9.5 Control procedures for development systems

Stages 1 to 6 will be basically the same except that an estimate on the effect of the project's timescale will need to be included in steps 3 and 4. Step 7 might not be required as system acceptance might not have taken place yet and acceptance testing of the changes will be included in that.

The release of software in step 8 will not be needed if the system is not yet operational, although master copies of products will need to be updated.

9.6 Reasons for scope creep

There can be several different system designs that meet the users' requirements and one may be selected which is more elaborate and involves more processing.

Essential housekeeping and security driven requirements might become apparent – for example, additional validation of the input to ensure that the database remains consistent.

Chapter 10

The problem for Amanda at IOE would be that the new maintenance group accounts subsystem would essentially be an extension to and enhancement of the existing maintenance accounting system, so that the interfacing of an off-the-shelf package might involve quite a few difficulties. This seems to indicate that bespoke development is needed. An alternative approach might be to consider replacing the whole of the maintenance accounting system with a new off-the-shelf application.

10.1 Choice of type of package at IOE

For the first 2000 FPs	$\$967 \times 2,000$	=	\$1,934,000
For the next 500 FPs	$\$1,019 \times 500$	=	\$509,500
For the next 500 FPs	$\$1,058 \times 500$	=	\$529,000
For the last 200 FPs	$\$1,094 \times 200$	=	\$218,800
for all 3,200 FPs			\$3,191,300

10.2 Calculation of charges for a project

For changed FPs	$500 \times 600 \times (150/100)$	=	\$450,000
For additional FPs	200×600	=	\$120,000
Total charge			\$570,000

10.3 Calculating the cost of additional functionality

The supplier will need to quote a price that will include a margin to cater for possible increases in equipment prices. It might turn out that actual prices do not increase as much as was estimated – in the case of IT equipment some prices are likely to go down – but the customer would still have to pay the additional margin. If the contract specifies a fixed charge plus the actual cost of materials and equipment, then the customer in this case would be better off.

10.4 Advantage to customer of variable cost charges

System X savings would be $\text{£}20 \times 20 \text{ hours} \times 4 \text{ years} = \text{£}1,600$, for the automatic scale point adjustment facility, and $\text{£}20 \times 12 \text{ hours} \times 2 \text{ times a year} \times 4 \text{ years} = \text{£}1,920$, for the bar chart production facility. In total the saving for System X would be £3,520.

10.5 Calculating value for money

For System Y, the saving would be £300 \times 0.5 (to take account of the probability of change). That is, £150.

Even though System X costs £500 more, it will still give better value for money. Note that discounted cash flow calculations could be applied to these figures.

10.6 Evaluation methods

- i. The usability of an existing system could be evaluated by such means as the examination of user handbooks, the observation of demonstrations and practical user trials.
- ii. This is clearly tricky. One would have to evaluate the methods that the developers intend to use to see whether they adhere to good interface design practice. One might also examine any interface standards that are in use by the supplier.
- iii. Note that the question focuses on the costs of maintenance, rather than that of reliability. The cost of unexpected maintenance could be reduced, at least for a short time, by passing this risk to the supplier if there is a comprehensive warranty. The warranties provided by suppliers would therefore need to be scrutinized. Discussion with reference sites might also be helpful.
- iv. Once again guarantees could be put in place by suppliers concerning this. The nature of these guarantees could be examined.
- v. Training materials could be examined. The training staff could be interviewed and their CVs examined. Reference sites who have already used the supplier's training services could be approached for their views.

Chapter 11

An analyst/programmer is expected to be able to carry out both analysis and programming tasks. It is likely, however, that the kinds of analysis tasks undertaken will be restricted. They may, for example, do the analysis work for enhancements to existing systems but not of completely new applications. Making this broad assumption, a list of tasks and responsibilities might be as follows:

- carry out detailed investigations of new requirements for existing computer applications;
- analyse the results of investigations and review the solutions to problems experienced, including the estimation of relevant costs;
- prepare systems specifications in accordance with organizational standards;
- conduct appropriate systems testing;
- prepare functional module specifications.;
- produce and modify module structure diagrams;
- code and amend software modules;
- carry out appropriate unit testing;
- produce and amend user documentation;
- liaise with users, carrying out appropriate training in the use of computer applications where required.

11.1 Tasks and responsibilities of an analyst/programmer

A problem here is that the programmers who make most use of reused components will, as a consequence, be producing less code themselves. You also want to encourage programmers to produce software components that other people can use: this might help the productivity of the organization but not that of the current project that they are working on!

11.2 Rewarding reuse

You need to have a method, like function point analysis, which measures the functions and features actually delivered to the user. You also need to have some way of measuring the code used in the application that has been taken from elsewhere. Percentage targets of the amount of reused code to new code could be set and staff rewarded if the targets are met. As an alternative, the savings made by reuse could be measured and a profit-sharing scheme could be operated.

Programmers could be encouraged to produce and publish reusable components by a system of royalties for each time a software component is reused.

11.3 Financial incentives for top executives

This exercise was designed to be thought-provoking. Some thoughts that have come out of discussion on this topic are given below.

- To some extent, material wants and, therefore, the motivation to obtain more money to satisfy these wants, can be generated through the marketing and advertising of new types of goods and services – but how likely is this to be at the very top?
- Large salaries are associated with status, esteem and success. It could be that these are the real reward.
- Historically, wealth has been associated with power, such as the ownership of land.

The essential point is that for many people money is not just a means of satisfying material wants.

11.4 High and low motivational incidents

This will obviously depend on individual experiences.

11.5 Social loafing

Among other ideas, the effects of social loafing can be reduced by:

- making the work of each performer individually identifiable;
- involving and interesting group members in the outcome of group efforts;
- rewarding individuals for their contributions to the group effort (rather like sports teams who pick out a 'club player of the year').

11.6 Effect of IT on the Delphi technique

Developments in IT that assist co-operative working, especially the advent of electronic mail and groupware such as Lotus Notes, will cut quite considerably the communication delays involved in the Delphi technique.

More than one type of power can be involved in each case.

- i. Some *expert* power is involved here, but for those who are subject to the audit, the main type of power is *connection* power as the auditor will produce a report that will go to higher management. External auditors often have *coercive* power.
- ii. Here, power will mainly be *expert-* and *information-based*, but as the consultant will report to higher management, *connection* power also exists.
- iii. This sounds pretty *coercive*.
- iv. Brigette has some *connection* power. The technical expertise that is involved in her job means she has some *expert* power. She has little or no *coercive* power as she is not the manager of the staff involved. She might be able to exert some *reward* power on the basis of an informal 'I'll do you a favour if you'll do me a favour' arrangement!
- v. Amanda is unlikely to have direct *coercive* powers although she might be able to institute disciplinary procedures. Through the system of annual reviews common to many organizations, she might have some *reward* power. *Connection* power, through her access to higher management, is also present. Her access to users means she has *information* power. If she brings specific expertise to the project (such as analysis skills) she might have some *expert* power. By acting as a role model that other project team members might want to emulate she may even be displaying *referent* power!

11.7 Classification of types of power

- i. The clerk will know much more than anyone else about the practical details of the work. Heavy *task-oriented* supervision would therefore not be appropriate. As the clerk is working in a new environment and forging new relationships, a considerable amount of people-oriented supervision/support might be needed initially.
- ii. Both task-oriented and people-oriented management would be needed with the trainee.
- iii. The experienced maintenance programmer has probably had considerable autonomy in the past. The extensions to the systems could have a considerable, detailed, impact on this person's work. A very carefully judged increase in task-oriented management will be required for a short time.

11.8 Appropriate management styles

Chapter 12

12.1 Selection of payroll package for college

- (a) Carry out an investigation to find out what the users' requirements really are. This might uncover that there are different sets of requirements for different groups of users.
- (b) Organize the requirements into groups relating to individual qualities and attributes. These might be, for example, functionality (the range of features that the software has), price, usability, capacity, efficiency, flexibility, reliability and serviceability.
- (c) Some of these requirements will be of an absolute nature. For example, an application will have to hold records for up to a certain maximum number of employees. If it cannot, it will have to be immediately eliminated from further consideration.
- (d) In other cases the requirement is relative. Some of the relative requirements are more important than others. A low price is desirable but more expensive software cannot be ruled out straightaway. This can be reflected by giving each of the requirements a rating, a score out of 10, say, for importance.
- (e) A range of possible candidate packages needs to be identified. If there are lots of possibilities, an initial screening, for instance, by price, can be applied to reduce the contenders to a manageable shortlist.
- (f) Practical ways of measuring the desired qualities in the software have to be devised. In some cases, for example with price and capacity, sales literature or a technical specification can be consulted. In other cases, efficiency for instance, practical trials could be conducted, while in yet other cases a survey of existing users might provide the information required.
- (g) It is likely that some software is going to be deficient in some ways, but that this will be compensated by other qualities. A simple way of combining the findings on different qualities is to give a mark out of 10 for the relative presence/absence of the quality. Each of these scores can be multiplied by a score out of 10 for the importance of the quality (see (d)) and the results of all these multiplications can be summed to give an overall score for the software.

12.2 Relationships between pairs of quality factors

- **Indifferent** Usability and reusability would seem to have little bearing on each other in spite of the similarity in their names. (Although it is usually possible to identify at least a tenuous complementary or conflicting relationship between two quality factors if you try hard enough).

- **Complementary** A program that demonstrated flexibility might also be expected to have a high degree of maintainability.
- **Conflicting** A program can be highly efficient because it exploits the architecture of a particular type of hardware to the full, but so not be easy to transfer to another hardware configuration.

The presence of the same software quality criterion for more than one software quality factor would indicate that the software quality factors are complementary.

12.3 Quality criteria

There are many that could be defined and just two examples are given below. One point that may emerge is that the software might be best broken down into a number of different function areas, each of which can be evaluated separately, such as document preparation, presentation, mail merging and so on. For example:

12.4 Possible quality specifications for word processing software

- **quality** – ease of learning;
- **definition** – the time taken, by a novice, to learn how to operate the package to produce a standard document;
- **scale** – hours;
- **test** – interview novices to ascertain their previous experience of word processing. Supply them with a machine, the software, a training manual and a standard document to set up. Time how long it takes them to learn how to set the document up.
- **worst** – 4 hours;
- **planned** – 2 hours;
- **best** – 1 hour;
- **now** – 4 hours.

or

- **quality** – ease of use;
- **definition** – the time taken for an experienced user to produce a standard document;
- **scale** – minutes;

- **test** – time user who has experience of package to produce the standard document;
- **worst** – 45 minutes;
- **planned** – 40 minutes;
- **best** – 35 minutes.

This topic of evaluation is an extensive one and the pointers above leave all sorts of unanswered questions in the air. Readers who wish to explore this area should read one of the more specialist books on the topic.

12.5 Availability and mean time between failures

Each day the system should be available from 18.00 – 8.00 hours, = 10 hours. Over four weeks that should be $10 \times 5 \times 4$ hours = 200 hours. It was unavailable for one day, i.e. 10 hours. It was unavailable until 10.00 on two other days, = 4 hours. The hours available were therefore $200 - 10 - 4 = 186$ hours. Availability would therefore be $186/200 \times 100 = 93\%$

Assuming that three failures are counted, *mean time between failures* would be $186/3 = 62$ hours.

12.6 Entry requirements for an activity different from the exit requirements for another activity that immediately precedes it

It is possible for one activity to start before the immediately preceding activity has been completely finished. In this case, the entry requirement for the following activity has been satisfied, even though the exit requirement of the preceding activity has not. For example, software modules could be used for performance testing of the hardware platform even though there are some residual defects concerning screen layouts.

Another situation is where the entry requirements could vary from the preceding exit requirements is where a particular resource needs to be available,

12.7 Entry and exit requirements

- **Entry requirements** A program design must have been produced that has been reviewed and any rework required by the review must have been carried out and been inspected by the chair of the review group.

- **Exit requirements** A program must have been produced that has been compiled and is free of compilation errors; the code must have been reviewed and any rework required by the review must have been carried out and been inspected by the chair of the review group.

It should be noted that the review group may use checklists for each type of product reviewed and these could be regarded as further entry/exit requirements.

- **Control of equipment** In the software environment, this would include the software tools involved. Of particular concern, for example, would be the compilers used. These would need to be known to work correctly and also to work in the same way as compilers of the same language in other machine environments in order to assist portability.
- **Testing status** There would be concern that software components which are being updated are not released to users before adequate testing has taken place. These issues are addressed by change control and configuration management procedures.
- **Distribution** Some concerns here would be to do with the security of physical copying and distribution of software via magnetic and electronic media. Configuration management would also be important to ensure that the version of the software that is being shipped has all the correct components.

12.8 Application of BS EN ISO 9001 standards to the software environment

The project manager could check who actually carried out the certification. They could also discover the scope of the BS EN ISO 9001 certification that was awarded. For example, it could be that certification only applied to the processes that created certain products and not others.

12.9 Precautionary steps when work is contracted out?

Perhaps the most important point is that the project manager will need to be reassured that the *specification* to which the contractors will be working is an adequate reflection of the requirements of the client organization.

The quality circle would be looking at the process in general while the review group would look at a particular instance of a product. The use of review groups alone could be inefficient because they could be removing the same type of defect again and again rather than addressing, as the quality circle does, the task of stopping the defects at their source.

12.11 The important differences between a quality circle and a review group?

Further reading

General introductory books on project management – not IS specific

Haynes, Marion E., *Project management : from idea to implementation*, Kogan Page better management skills, London, Kogan Page, 1990.

Haynes, Marion E., *Project management*, rev. edn, A fifty-minute series book, Menlo Park, Calif., Crisp Publications, 1996.

Weiss, Joseph W., and Robert K. Wysocki, *5-phase project management : a practical planning & implementation guide*, Reading, Mass., Addison-Wesley, 1992.

Nickson, David, and Suzy Siddons, *Managing projects*, Oxford, Made Simple Books; Butterworth-Heinemann, 1997.

General books on software and IS project management

Bennatan, E. M., *Software project management : a practitioner's approach*, 2nd edn, London; New York, McGraw-Hill, 1995.

Yeates, Donald, and James Cadle, *Project management for information systems*, 2nd edn, London, Pitman Publications, 1996.

Ince, Darrel, Helen Sharp, and Mark Woodman, *Introduction to software project management and quality assurance*, The McGraw-Hill international series in software engineering, London; New York, McGraw-Hill, 1993.

Other books worth looking at

Brooks, Frederick P., *The mythical man-month : essays on software engineering*, anniversary edn, Reading, Mass., Addison-Wesley, 1995.

The classic exposition of the central issues of software project management from the man who was in charge of the IBM 360 Operating System development project. You should try to look at it at some time.

Kemerer, Chris F. (ed.), *Software project management : readings and cases*, Chicago, Irwin, McGraw-Hill, 1997.

A collection of classic papers on topics such as estimation, risk management, life cycles, re-use and process improvement. Strongly recommended.

Whitten, Neal, *Managing software development projects : formula for success*, New York; Chichester, Wiley, 1995.

Strategic planning and scope

Clare, Chris, and Gordon Stuteley, *Information systems – strategy to design*, Tutorial guides in computing and information systems; 7, London, Chapman & Hall, 1995.

Vonk, Roland, *Prototyping : the effective use of CASE technology*, New York, Prentice Hall International, 1990.

Gilb, Tom, and Susannah Finzi, *Principles of software engineering management*, Wokingham, England; Reading, Mass., Addison-Wesley, 1988.

Ould, Martyn, *Strategies for software engineering : the management of risk and quality*, Wiley series in software engineering practice, New York; Chichester, Wiley, 1990.

Planning

Bradley, Ken, *PRINCE: A Practical Handbook*, Oxford; Boston, Butterworth-Heinemann, 1993.

Bentley, Colin, *Prince 2 : a practical handbook*, Computer weekly professional series, Oxford; Boston, Butterworth-Heinemann, 1997.

Harrison, F. L., *Advanced project management : a structured approach*, Aldershot, Gower, 1992.

A general project management book – not just software projects.

Lock, Dennis, *Project management*, 6th edn, Brookfield, Vt., Gower, 1996.
An alternative to the Harrison book, it shows some awareness of PRINCE.

Estimation

Boehm, Barry W., *Software engineering economics*, Prentice-Hall advances in computing science and technology series, Englewood Cliffs, N.J., Prentice-Hall, 1981.

Along with the Brooks book one of the most frequently cited books on software project management.

Symons, Charles R., *Software sizing and estimating : Mk II FPA (function point analysis)*, Wiley series in software engineering practice, New York; Chichester, Wiley, 1991.

A book by the inventor of Mark II function points.

DeMarco, Tom, *Controlling software projects : management, measurement & estimation*, New York, Yourdon Press, 1982.

Control and configuration management

Youll, David P., *Making software development visible : effective project control*, Wiley series in software engineering practice, New York; Chichester, John Wiley, 1990.

Central Computer and Telecommunications Agency, *Configuration management, IT infrastructure library*, London, HMSO, 1990.

CCTA, *Configuration Management Guide*, CCTA, 1989.

Buckle, J. K., *Software configuration management*, London, Macmillan, 1982.

Risk

Boehm, Barry W., *Software risk management*, Washington, DC, IEEE Computer Society Press, 1989.

Grey, Stephen, *Practical risk assessment for project management*, Wiley series in software engineering practice, New York; Chichester, Wiley, 1995.

Down, Alex, Michael Coleman and Peter Absolon, *Risk management for software projects*, IBM McGraw-Hill series, London; New York, McGraw-Hill, 1994.

Charette, Robert N., *Software engineering risk analysis and management*, McGraw-Hill software engineering series, New York, Intertext Publications; McGraw-Hill Book, 1989.

Quality and testing

Humphrey, Watts S., *Managing the software process*, repr. 1990 with corrections, SEI series in software engineering, Reading, Mass., Addison-Wesley, 1990.

Manns, Tom, and Michael Coleman, *Software Quality Assurance*, 2nd edn, Macmillan, 1996.

Manns, Tom, and Michael Coleman, *Software quality assurance*, Macmillan computer science series, Hounds Mills, Basingstoke, Port Washington, NY., Macmillan Education; distributed by Scholium International, 1988.

Daily, K., *Quality management for software*, Manchester, NCC Blackwell, 1992.

People Management

DeMarco, Tom, and Timothy R. Lister, *Peopleware : productive projects and teams*, 2nd edn, New York, Dorset House, 1999.

Healy, Patrick L., *Project management : getting the job done on time and in budget*, Port Melbourne, Vic.; Newton, USA, Butterworth-Heinemann, 1997.

Although it presents itself as a general book on project management, it is relatively thin on the technical aspects of project planning. However, it makes many useful points about dealing with political and behavioural issues.

Charles B. Handy, *Understanding Organizations*, 4th edn, London, Penguin, 1993.

A general book on this topic which is not IS-specific.

Belbin, R. Meredith, *Management teams : why they succeed or fail*, Oxford; Boston, Butterworth-Heinemann, 1996.

Belbin, R. Meredith, *Team Roles at Work*, Oxford; Boston, Butterworth-Heinemann, 1993.

This updates but does not replace the 1981 book

Project management standards

British Standards Institution, *BS 6079 Guide to project management*, London, BSI, 1996.

Central Computer and Telecommunications Agency, *PRINCE 2 : [project management for business]*, London, The Stationery Office, 1996.

Project Management Institute and PMI Standards Committee, *A Guide to the project management body of knowledge*, Upper Darby, PA: Project Management Institute, 1996: obtainable via WWW from <http://www.pmi.org>.

Contracts

David Bainbridge, *Introduction to computer law*, 3rd edn, London, Pitman, 1996.

Index

A

Absolon, P, *see* Down, R. A., Absolon, P. and Coleman, M.
acceptance
 by customer 208
 of contracted products 308
acceptance criteria and ISO 12207 313
accounting rate of return, *see* return on investment
accumulated cost chart 166
acquisition goal definition in
 Euromethod 292–293
acquisition management 289, 292–295
acquisition planning in
 Euromethod 293–295
acquisition process and ISO 12207 304, 305–308
acquisition strategy in
 Euromethod 294–295
activities 266
 identification of 111
 subdivision of 187
activity plan, ideal 30–31, 109–110
activity planning 107–132
 objectives of 108
 reasons for 107
activity schedule 151
activity span 130
activity standard deviation in
 PERT 145–146
actual cost of work performed 184
ACWP, *see* actual cost of work performed
adaptation plan 292
 planning 295–300
 strategy 297–300
adaptations in Euromethod 290, 292
additive tasks 223
AIPM, *see* Australian Institute of Project Management

Albrecht, A. J. and Gaffney, J. E. on function points 89
algorithmic models, *see* parametric models
alternative dispute resolution 206
ami, *see* Application of Metrics in Industry
analogy in effort estimation 88–89
ANGEL 88–89
APM, *see* Association for Project Management
Application of Metrics in Industry (ami) 258
arbitration 206
ARR, *see* return on investment
assessment criteria definition in product quality 244
Association for Project Management 320–321
attribute specifications, *see* quality specifications
audits and ISO 12207 311
Australian Institute of Project Management 319
autonomy as job characteristic 220
availability 245

B

backward pass in CPM 123, 125–127
Bainbridge, D. on specifications in contracts 198
ball charts 176–177
Banker, R. D., Kauffman, R. and Kumar, R. on object points 94
bar charts 154
baseline budget 181–182
Basili, V. R. and Rombach, H-D. on Goal/Question/Metric approach 257
BCS, *see* British Computer Society

- BCWP, *see* budgeted cost of work performed
 BCWS, *see* budgeted cost of work scheduled
 Belbin, R. Meredith
 management teams 222
 staff eligibility and suitability 215
 Bennington, H. D. on the waterfall process model 65
 bespoke 202
 bespoke packages 275
 bespoke systems 193
 Boehm, B. W.
 classification of estimating methods 85
 COCOMO 98
 estimates as goals 84
 generic risks 142–143
 risk engineering 135
 spiral model 67
 staff productivity 215
 Theory W 14
 Bootstrap 251
 bottom-up estimating 86
 brainstorming 256–257
 British Computer Society 324
 British Standards Institution 251, 281
 Brooks' law 82
 Brooks, F. P.
 chief programmer teams 231
 intangibility of software 237
 nature of software 79
 ‘No silver bullet’ 3
 software versus other engineered artefacts 3
 BS 5750, *see* BS EN ISO 9001:1994
 BS 6079 111, 138, 281–288
 and planning 285
 and project control 287
 and project organization 283–285
 definition of project 37
 planning steps 286
 project life cycle 282–283
 supporting techniques 287–288
 BS EN ISO 9001:1994 61, 188, 248–250
 budget variance 184
 budgeted cost of work performed 180
 budgeted cost of work scheduled 180, 183
 Burman, P. J. on priority listing 158
 case-based reasoning, *see* analogy in effort estimation
 cash flow and incremental delivery 73
 cash flow forecasting 42–43
 CCTA, *see* Central Computer and Telecommunications Agency
 Central Computer and Telecommunications Agency 19, 92, 201, 269, 281
 change control 188–190, 207
 and prototyping 72–73
 in BS 6079 287
 procedures 189
 changed requirements and contract pricing 196
 Checkland, P. and Scholes, P. on soft systems 59
 checkpoints 31, 172
 in PRINCE 2 173
 Cheney, P. M.
 influences on productivity 220
 programming productivity 215
 chief programmer teams 231–232
 clarification of supplier proposals 202
 clean rooms in joint application development 64
 clean-room development 254–255
 closed systems 7
 CMM, *see* Capability Maturity Model
 co-ordination 108–109
 COCOMO 97–103
 basic model 97
 COCOMO 81 97
 development effort multipliers 98
 intermediate cost drivers 99
 intermediate model 98
 see also COCOMO II
 COCOMO II 100
 application composition model 100
 early design model 100
 early design model 101
 effort multipliers 102–103
 exponent drivers
 architecture/risk resolution 101
 development flexibility 101
 precededness 101
 process maturity 102
 team cohesion 101
 post architectural model 101
 scale factors 101
 Coleman, M., *see* Down, R. A., Absolon, P. and Coleman, M.
 COMET 72

command groups 221
compensatory tasks 223
complexity as risk driver 297
configuration librarian, duties of 188
Configuration Librarian in PRINCE 2 272
configuration manager, *see configuration librarian*
conjunctive tasks 224
constraints 265
construction approach in Euromethod 298
 in Euromethod 299
Constructive Cost Model, *see COCOMO and COCOMO II*
contingency 155
contingency measures 33, 267
contract management 191–209
 change control 207
 decision points 206
 joint reviews 207
 supplier defaults 206–207
contract monitoring 295
contract placement 198–203
contract preparation and
 ISO 12207 306–308
contract terms
 acceptance procedures 205
 customer commitments 205
 definitions and terminology 204
 environment 205
 form of agreement 204
 goods and services to be supplied 204
 ownership of software 204–205
 price and payment method 206
 project and quality management 205
 standards 205
 timetable 206
contractor selection 197
contracts
 fixed price 193
 fixed price per unit 193
 goods versus services 192–196
 terms of 203–206
 time and materials 193
 types of 192–193
control
 and ISO 12207 308–309
 cycles 11–12
 data versus information 11
control points 173
control systems, hierarchical 15
copyright of software 204
correctness 238

cost benefit analysis 40–42
 assessable indirect benefits 41
 development costs 41
 direct benefits 41
 evaluation techniques 43
 in Euromethod 293
 intangible benefits 41
 net profit 43
 operating costs 41
 set-up costs 41
cost charts 180–183
cost monitoring 180
cost performance index 184
cost profile of projects 165
cost schedule 151, 164
cost variance 184
costing 108
costs
 overheads 164
 staff 164
 usage charges 165
COTS, *see customized off-the-shelf software*
Couger, J. D and Zawacki, R. A.
 job characteristics 220
 job satisfaction of programmers 221
 social needs of programmers 215
CPI, *see cost performance index*
CPM, *see critical path method*
critical path 123, 129, 137, 185
 and resource allocation 159
 identifying 127
 in CPM 123
 shortening 186–187
critical path method 116, 117–129
 earliest dates 123
 latest dates 123
 slack 123
criticality 237
cumulative expenditure charts 179–180
 and revised schedules 180
curriculum vitae, *see CV*
customized off-the-shelf software 193, 202
CV 215, 216

D

dangles in CPM 119
day rates 217–218
DCF, *see discounted cash flow*

- Dearnley, P. A., *see* Mayhew, P. J., Worseley, C. J. and Dearnley, P.A.
- decision making 224–226
in groups 225
mental obstacles to 225–226
- decision points in Euromethod 291, 300
- decisions, structured or unstructured 224
- decommissioning 280
costs 42
- decision points in Euromethod 295
- decision trees 52–54
- deliverables, *see* products
- Delphi technique 226
- demonstrations 202
- departmentalization 230
- dependent activities 266
- description approach in Euromethod 298
- design to cost 85
- development and ISO 12207 305, 309–310
- differentiation of staff roles 230
- Dijkstra, E. W. on structured
programming 254
- discounted cash flow 153
- disjunctive tasks 223
- documentation
and ISO 12207 303
characteristics of 304
generic types 304
purpose of 303–304
- Dooley, Adrian on BS 6079 281, 282
- Down, R. A., Absolon, P. and Coleman, M.
on risk reporting 174
- dummy activities 121
and lagged activities in CPM 122
- Duncan, W. R. on Project Management
Institute 316
- duration
removing bottlenecks 129
shortening by parallelism 129
shortening of project 129
- E**
-
- earliest start dates 154, 156–157
- earned value monitoring 182
- earned value analysis 180–185
and BS 6079 287, 288
- effectiveness, *see* efficiency
- efficiency 16, 238
in ISO 9126 241
- sub-characteristics of 242
- effort estimation
complexity assessment 85
difficulties with 79–80
evaluation of tenders 82
measurement of work 84
need for historical data 84
over and under-estimates 82
political implications 81
procedural code approach 96–97
subjective nature of 81
timing of 81–82
- egoless programming 231
- elapsed time 31, 32
- End Stage Assessments in PRINCE 2 173
- entry requirements 246
- equipment as a resource 153
- escalation 278
of commitment 225
- escrow 205
- estimates 28
bottom-up 32
in small projects 266
recording accuracy of 135
top-down 32
- Euclidean distance 88
- Euromethod 197, 206, 282, 289–301
and process models 76–77
distinction between uncertainty and
complexity 61
- Euromethod views of information systems
business information 301
business processes 301
computer system architecture 301
computer system data 301
computer system functions 301
work practice 301
- European Union 197, 205
- EVA, *see* earned value analysis
- evaluation of proposals 202
- evolutionary delivery and ISO 12207 310
- evolutionary development 299
- Exception Plan in PRINCE 2 273
- Exception Report in PRINCE 2 273, 278
- Executive role in PRINCE 2 272
- exit requirements 34, 247
- expectancy theory of motivation 219
- expert judgement in effort
estimation 87–88
- expert systems, *see* systems,
knowledge-based
- extendibility measurement 245

- Dearnley, P. A., *see* Mayhew, P. J., Worseley, C. J. and Dearnley, P.A.
- decision making 224–226
in groups 225
mental obstacles to 225–226
- decision points in Euromethod 291, 300
- decisions, structured or unstructured 224
- decommissioning 280
costs 42
- decision points in Euromethod 295
- decision trees 52–54
- deliverables, *see* products
- Delphi technique 226
- demonstrations 202
- departmentalization 230
- dependent activities 266
- description approach in Euromethod 298
- design to cost 85
- development and ISO 12207 305, 309–310
- differentiation of staff roles 230
- Dijkstra, E. W. on structured
programming 254
- discounted cash flow 153
- disjunctive tasks 223
- documentation
and ISO 12207 303
characteristics of 304
generic types 304
purpose of 303–304
- Dooley, Adrian on BS 6079 281, 282
- Down, R. A., Absolon, P. and Coleman, M.
on risk reporting 174
- dummy activities 121
and lagged activities in CPM 122
- Duncan, W. R. on Project Management
Institute 316
- duration
removing bottlenecks 129
shortening by parallelism 129
shortening of project 129
- E**
-
- earliest start dates 154, 156–157
- earned value monitoring 182
- earned value analysis 180–185
and BS 6079 287, 288
- effectiveness, *see* efficiency
- efficiency 16, 238
in ISO 9126 241
- sub-characteristics of 242
- effort estimation
complexity assessment 85
difficulties with 79–80
evaluation of tenders 82
measurement of work 84
need for historical data 84
over and under-estimates 82
political implications 81
procedural code approach 96–97
subjective nature of 81
timing of 81–82
- egoless programming 231
- elapsed time 31, 32
- End Stage Assessments in PRINCE 2 173
- entry requirements 246
- equipment as a resource 153
- escalation 278
of commitment 225
- escrow 205
- estimates 28
bottom-up 32
in small projects 266
recording accuracy of 135
top-down 32
- Euclidean distance 88
- Euromethod 197, 206, 282, 289–301
and process models 76–77
distinction between uncertainty and
complexity 61
- Euromethod views of information systems
business information 301
business processes 301
computer system architecture 301
computer system data 301
computer system functions 301
work practice 301
- European Union 197, 205
- EVA, *see* earned value analysis
- evaluation of proposals 202
- evolutionary delivery and ISO 12207 310
- evolutionary development 299
- Exception Plan in PRINCE 2 273
- Exception Report in PRINCE 2 273, 278
- Executive role in PRINCE 2 272
- exit requirements 34, 247
- expectancy theory of motivation 219
- expert judgement in effort
estimation 87–88
- expert systems, *see* systems,
knowledge-based
- extendibility measurement 245

Herzberg two-factor theory on job satisfaction 219
 Herzberg, F.
 job satisfaction 219
 job enlargement and enrichment 221
 Highlight Report in PRINCE 2 277–278
 Hughes, R. T. on expert judgement 88
 Humphrey, Watts on Capability Maturity Model 251
 hygiene factors 219

I

IBM 114, 223, 253
 ideal activity plan, modification of 151
 IEEE, *see* Institute of Electrical and Electronics Engineers
 IEEE 12207, *see* ISO 12207
 IFPUG, *see* International Function Point User Group
 implementation requirements 247
 incremental delivery 62, 73–76
 advantages of 73–74
 and ISO 12207 310
 disadvantages of 74
 open technology plan 75–76
 plan 293
 planning of increments 75
 incremental development 299
 increments 263
 induction of new staff 217
 information
 operational 16–17
 strategic 16–17
 tactical 16–17
 Information Engineering 14, 62
 Information Systems Examination Board 324–325
 initial and final states in Euromethod 290
 inspections 217, 253
 installation approach in Euromethod 298, 299
 Institute of Electrical and Electronics Engineers 304
 insurance to cover supplier's liability 206
 intermediate products, errors in 246
 internal rate of return 47–49

International Function Point User Group 90
 interoperability 239
 interviews 216
 invitation to tender 197, 201
 IRR, *see* internal rate of return
 ISEB, *see* Information Systems Examination Board
 ISO 12207 201, 205, 207–206, 303–313
 and process models 76–77
 tailoring of standard 306
 ISO 9001, *see* BS EN ISO 9001:1994
 ISO 9001:1994, *see* BS EN ISO 9001:1994
 ISO 9126 200, 241–245

J

Jackson structured programming 59
 JAD, *see* joint application development
 Jensen, M. A., *see* Tuckman, B.W. and Jensen, M. A.
 job
 design 221
 enrichment 221
 holder profiles 216
 specifications 216
 joint application development 64, 298
 joint reviews and ISO 12207 311
 JSP, *see* Jackson structured programming

K

Kauffman, R. and Kumar, R.
 object points 95
 see also Bunker, R. D., Kauffman, R. and Kumar, R.
 key process areas 251
 Kitchenham, B. A. and Taylor, N. R. on software effort estimation 80
 KLOC 17
 see also source lines of code
 knowledge-based systems, *see* systems, knowledge-based
 KPA, *see* key process areas
 Kumar, R., *see* Kauffman, R. and Kumar, R.
 and Bunker, R. D., Kauffman, R. and Kumar, R.

L

- labour as a resource 153
 ladder of activities 122
 lag in precedence networks 130
 lagged activities in CPM 122
 Lawrie, Robyn, *see* Radford, Paul and Lawrie, Robyn
 leadership 226–229
 autocratic or democratic 228
 directive or permissive 228
 styles 227–228
 least squares regression 94
 Lessons Learnt Report in PRINCE 2 280
 licence to use software 193
 life cycle, *see* process models
 Likert, Rensis on leadership styles 228
 lines of code 195, 196
 liquidated damages 206, 207
 logical internal file types, function point complexity rating 90
 in function point analysis 89

M

- Madnick, S. E., *see* Hamid, T. K.
 maintainability 238
 in ISO 9126 241
 measurement of 245
 sub-characteristics of 242
 maintenance and ISO 12207 305
 make versus buy 306, 308
 management, activities involved in 8–9
 task-oriented or people-oriented 228
 Mark II function points 92–94
 Martinez, Demian on product versus supplier criteria 199
 Maslow, Abraham on the hierarchy of needs 218
 materials as a resource 153
 matrix management structure 283–284
 Mayhew, P. J., Worseley, C. J. and Dearnley, P.A. on prototyping 72
 Mayo, Elton on productivity 214
 McCall, James A. on software product quality 238–240
 McGregor, Donald on Theory X and Y 214
 mean time between failures 245
 measurement 16
 of quality 240

measures

- of effectiveness 12, 22
 see also objectives
 performance 17
 predictive 17
 memorandum of agreement 201
 methods and technology 265–266
 milestones 31, 285, 300
 see also checkpoints
 Mills, Harlan on clean-room development 254–255
 MITP (Managing the Implementation of the Total Project) 114–115
 MoA, *see* memorandum of agreement
 modularity 17
 money as a resource 153
 monitoring
 of supplier 308
 use of schedule in 107–108
 monitoring and control 169–190
 monitoring priorities 185–186
 activities using critical resources 186
 activities with high risks 186
 activities with no free float 185
 activities with small float 186
 critical path activities 185
 monitoring progress, in BS 6079 287
 Monte Carlo simulation 52–53, 149–150, 286
 most probable error lists 257
 motivation 217–221
 and estimates 84
 and financial reward 214
 and realistic targets 84
 and the hierarchy of needs 218
 and the Taylorist model 217–218
 by setting targets 108
 improvement of 221
 motivators 219

N

- n*-version development 237
 NACCB, *see* National Council for Certification Bodies
 National Computing Centre and escrow services 205
 National Council for Certification Bodies 251

National Vocational Qualifications 322–323
near-critical paths 129
negotiated procedure in tendering 198
net present value 45–47, 288
used with decision trees 53
net profit 43–44
network models, formulation of 117–120
network planning models 116
Newman, Pippa on PRINCE 2 and British standards 281
nodes
in CPM 117, 118
intermediate 118
sink 118
source 118
notification of proposal acceptance 203
NPV, *see* net present value
NVQ, *see* National Vocational Qualifications

O

OB, *see* organizational behaviour
object-oriented development 59
object points 94–96
objectives 6, 12, 22, 265, 275
in incremental delivery 74
Off-Specification in PRINCE 2 277
off-the-shelf
applications 193
packages 275
software 202
Oldham-Hackman job characteristics model 220
once-through, *see* waterfall model
and ISO 12207 310
one-shot approach 299
see also waterfall model
open systems 7
open technology plan in incremental delivery 74
open tendering process 197
operation and ISO 12207 304
operational requirement 198
OR, *see* operational requirement
organizational behaviour 213
organizational structures 229–232
centralized versus decentralized 231
formal and informal 229
functional versus task-oriented 230

hierarchical approach 229
staff and line workers 229
Orlikowski, Wanda J. on team working 218
Ould, Martyn on technical planning 57
overload, information 225
overtime 187
ownership of software 204–205

P

parametric models 86–87
categorized as size or productivity models 87
Park, R. E. on definition of source lines of code 84
‘Parkinson’ estimation method 85
Parkinson’s law 82
partial completion reporting 173
payback period 44
PBS, *see* product breakdown structure
peer reviews 231
people management 211–234
PERT *see* programme evaluation review technique
Petri nets 61
PFD, *see* product flow diagram
Phillips, Dwayne on risk 136
PID, *see* Project Initiation Document
piece rates 217–218
plan
contents of 264–267
when to create 109
plan scenarios in Euromethod 293–294
PMBOK, *see* Project Management Body of Knowledge
PMI, *see* Project Management Institute
portability 239
in ISO 9126 241
sub-characteristics of 243
post implementation reviews 179
post-conditions 255
power
categorization as position or personal 227
coercive 227
connection 227
expert 227
information 227
legitimate 227
referent 227
reward 227

- pre-conditions 255
pre-requisites for activities 266
precedence networks 116, 130–131, 154
precedence requirements 111
 revision of 187–188
precededness 101
precedents in CPM 119
preferred vendors list 287–288
preventive measures in risk analysis 267
price to win 85
PRINCE 2 19, 112, 138, 139, 171, 173,
 269–280, 306, 315
 and BS 6079 281–282, 283
closing a project 280
controlling a stage 276
directing the project 275
initiating the project 275
managing product delivery 278
managing stage boundaries 279
project organization 271–272
project procedures 273–280
staged approach 4
 starting up a project 275
prioritization of activities 158
 by duration 158
 by total float 158
process models
 choice of 63–64
 prototyping 67–73
 selection of 76
 spiral 67
 V-process 66
 waterfall 65–66
process requirements 246–247
processes in project life cycle
 coding 5
 design 5
 execution 4
 feasibility study 3
 implementation 5
 installation 5
 maintenance and support 5
 planning 4
 requirements analysis 4
 specification 5
 validation 5
 verification 5
procurement 287, 295
 in Euromethod 291
product breakdown structure 28–29,
 112–113, 285
and work breakdown structures 283
product descriptions 28
product flow diagram 29–30, 112–114, 188
product qualities
 operation 238
 revision 238
 transition 238
product versus process quality 246
product-based work breakdown
 structure 285
products 6, 28, 237, 266
 management 28
 quality 28
 technical 28
programme evaluation review
 technique 116, 143–149, 186
 activity standard deviations 145–146
 advantages of 149
 calculation of z values 146, 147–148
 converting z values to probabilities 148
 event label conventions 145
 event standard deviations 147
 expected durations 144
 likelihood of meeting targets 146–148
 most likely time 144
 optimistic time 144
 pessimistic time 144
 risk assessment 143
programme management 38–39, 108, 275,
 292
 and feasibility studies 4
 programme director 39
programming productivity 215
progress assessment 171–172
progress data collection 173–175
project
 as system 7
 authority 12, 23
 definition of 2
 embedded systems 6
 evaluation 20, 37–56
 key characteristics of 2
project analysis 58
Project Approach in PRINCE 2 275
project approach selection 59–78
project assessment
 strategic 38–40
 technical 40
project assurance 171
 in PRINCE 2 272
Project Board 171
 in PRINCE 2 272
Project Brief in PRINCE 2 275

project control 169–173
 in Euromethod 300
 cycle 170
 project duration, trade-off against cost 161
Project End Report in PRINCE 2 280
Project Initiation Document 276
Project Issues in PRINCE 2 277
 project librarian, *see* configuration librarian
 Project Management Body of Knowledge
APM, see Association for Project Management
PMI 316–319
 Project Management Institute 316–319
 Project Management Team in PRINCE 2 272
 Project Manager in PRINCE 2 272, 276–277
 Project Mandate in PRINCE 2 275
 Project Plan in PRINCE 2 273, 276
 Project Quality Plan in PRINCE 2 275–276
 project steering committee 171
 project support in PRINCE 2 272–273
 project tolerances 273
 projects
 and activities 111
 objectives versus product-driven 27, 59
 prototypes
 as vehicle for learning 70
 evolutionary 67
 incremental 68
 mock-ups 70
 of functionality 71
 of interface 70
 partial: horizontal 70
 partial: vertical 70
 simulated interaction 70
 throw-away 67
 tools to support 71
 prototyping 62, 67–73, 142
 purchase order documents 287
 Pyster on software project management problems 9

Q

QMS, see quality management systems
 quality
 criteria 28
 procedures 25

reviews 34
 quality checks 266
 quality circles 255–257
 quality management systems 248–250
 quality metrics selection 243–244
 quality plan 138
 quality requirements 199
 quality specifications 237, 240

R

RAD, *see* rapid application development
 Radford, Paul and Lawrie, Robyn on payment of suppliers 193
 rapid application development 64–65
 ratings level definition in product quality 243
 rational economic model (of decision making) 224
 Raven, B. H., *see* French, J. R. P. and Raven, B. H.
 recruitment 215–217
 recruitment costs 155
 references 216
 reliability 238
 in ISO 9126 241
 measurement of 245
 sub-characteristics of 242
 reporting methods 171–172
 representations of suppliers 202
Request for Change in PRINCE 2 277
 request for proposal 306
 preparation of response 308
 requirements
 desirable 199
 exit 34
 functional 14–15
 mandatory 199
 modifications to 188
 quality 14
 resource 15
 system or software in ISO 12207 306
 users' concerning implementation 61
 requirements analysis in contract placement 198
 reserve, management 287
 resource allocation 33, 108, 110, 151–168
 additional staff, effect of 187
 and trade-offs 166
 staff productivity, consideration of 187

resource constraints 33–34, 109
resource histograms 154–157
resource requirements, identification of 153–154
resource schedule 151
 publication of 162–165
resource scheduling 154–159
resource smoothing 156–159
 and resequencing activities 158
resources 266–267
 nature of 152–153
restricted tendering process 198
résumé, *see* CV
return on investment 44–45
reusability 239
review points 173
reviews 309
RFP, *see* request for proposal
risk
 activity-based 32–33
 control 136
 directing 136
 engineering 135
 estimation 136
 evaluation 136
 generic or specific 137
 high level 27, 60
 importance of 133
 likelihood 139–140
 management 133–150
 monitoring 136
 planning 136
 premium 49, 50
 prioritization 140–141
 reduction 33, 142–143
 reporting 174
 staffing 136
risk analysis 110, 139–142, 267, 295–297
 in Euromethod 294
risk assessment and BS 6079 287
risk evaluation 50
 and cost benefit analysis 50–51
 and net present value 50
 decision trees 52–54
 identification and ranking 50
 risk matrix 50–51
 risk profile analysis 51
risk identification 135, 136, 137
 application factors 137
 changeover factors 138
 checklists 137
 environmental factors 138

expert systems, *see* systems,
 knowledge-based
hardware/software factors 138
health and safety 138
positive attitude to 136
project factors 138
project methods 138
staff factors 137
supplier factors 138
risk impact
 compromised quality or
 functionality 140
 costs of delay 140
 resource costs 140
Risk Log 276
risk management 135
risk reduction 142–143
 contingency planning 142
 hazard prevention 142
 likelihood reduction 142
 risk avoidance 142
 risk transfer 142
risk reduction leverage 141–142
risk, caused by
 assumptions 134, 135
 estimation difficulty 134, 135
 eventualities 134, 135
risks to schedule 143–147
risky shift 226, 231
ROI, *see* return on investment
Rombach, H-D, *see* Basili, V. R. and
 Rombach, H-D.
Ross, R. on Theory W 14

S

SADT 14
satisficing model (of decision making) 224
schedule
 activity 151
 cost 151
 resource 151
schedule performance index 184
schedule production 110
schedule variance 184
schedules 109
Schofield, C., *see* Shepperd, M. and
 Schofield, C.
Scholes, J., *see* Checkland, P. and Scholes, P.
scope creep 189

- Scottish Vocational Qualifications 322–323
- SEI, *see* Software Engineering Institute
- self-actualization 218
- sensitivity analysis 51–52
- sequencing and scheduling, differences between 115–116
- services as a resource 153
- Shepperd, M. and Schofield, C. on analogy 88
- shrink-wrapped software 193
- site visits 203
- situational factors 296
- skill variety, in job design 220
- slack 127
- Slater, C., *see* Goodland, M. and Slater, C.
- slip charts 176
- small projects 261–267
- SLOC, *see* source lines of code
- social loafing 224
- sociotechnical systems 7
- soft systems 59, 62
- software
- embedded 60
 - general or application-specific 59, 62
- software breakage 74
- software components and ISO 12207 310
- software effort estimation 79–106
- Software Engineering Institute (Carnegie Mellon University) 251
- software items and ISO 12207 310
- software life cycle data 303
- software product certification 252
- software projects, problems with 9–10
- software quality 235–259
- criteria 239–241
 - enhancement 252–258
 - measures 245
- software quality circles 256
- software quality definition 237–244
- software quality plan 246
- software tools, *see* tools
- software units and ISO 12207 310
- source lines of code 17, 80, 84, 87, 96–97
- and function points 91
- SOW, *see* statement of work
- space as a resource 153
- span of control 229
- SPI, *see* schedule performance index
- spiral model 67
- sponsor, project 283
- SSADM 27, 59, 61, 62, 64, 72, 92, 113–114, 138, 269, 289, 301
- staff allocation 161–162
- activity risk 161
 - and training 161–162
 - availability 161
 - task criticality 161
 - team building 162
- staff familiarization 155
- staff selection 215–217
- Stage Plan in PRINCE 2 273, 277
- stages 31, 35
- Stages in PRINCE 2 272
- stakeholder analysis in Euromethod 293
- stakeholders 13–14, 23–24
- standard normal deviates, *see* PERT, calculation of *z* values
- standards 28, 199, 281
- change control 25
 - configuration management 25
 - hardware 24
 - planning and control 25
 - quality, *see* quality procedures
 - software 24
- statement of work 283, 285
- Step Wise
- and student projects 261
 - approach to planning 19–35
 - planning activities 21
 - Step 0: Select project 20, 37
 - Step 1: Identify project scope and objectives 22–24
 - Step 2: Identify project infrastructure 24–26, 212, 236
 - Step 3: Analyse project characteristics 27–28, 58, 82, 134, 236
 - Step 4: Identify products and activities 28–31, 109–110, 212, 237
 - Step 5: Estimate effort for activity 32, 82, 109–110
 - Step 6: Identify activity risks 32–33, 134, 212
 - Step 7: Allocate resources 33, 152, 212
 - Step 8: Review and publicize the plan 34–35, 237
 - Step 9: Execute plan 35
 - Step 10: Lower level planning 35
- strategic plan 39
- strategic planning 20, 24

Stretton, A. on Australian Institute of Project Management 324
structured methods 64
structured programming 254
student projects, problems with 261–264
 incomplete systems 263
 uncertain design requirements 262
 unfamiliar tools 262
 users' lack of commitment 264
sub-objectives, *see* goals
sub-optimization 7
Supplier role in PRINCE 2 271–272
supply and ISO 12207 304
support activity 245
supporting processes and ISO 12207 305, 306
SVQ, *see* Scottish Vocational Qualifications
SWQC, *see* software quality circles
Symons, C. R. on Mark II function points 92
systems
 concurrent processing 60
 control 61
 control or data-orientated 59
 graphics-based 62
 knowledge-based 59, 60, 62, 137
 organic, semi-detached or embedded 98
 safety-critical 60, 62

T

targets, getting back on 186–188
task catalogue 112
task definitions 112
task groups 221
task identity 220
task owners in BS 6079 284
task significance 220
Taylor, Frederick 213, 217
Taylor, N. R., *see* Kitchenham, B. A. and Taylor, N. R.
Taylorism 213
TCA, *see* technical complexity adjustment
team formation 221–222
team leaders 171
Team Manager in PRINCE 2 278
team organization 26
Team Plan 278

technical complexity adjustment 92
 in function points 91
 in Mark II function points 93–94
technical plan, contents of 63
technical planning 57
tendering 295
tendering process
 negotiated 198
 open 197–198
 restricted 198
testability 238
Thamhain, H. J. on management tasks 8
Thayer on software project management problems 9
Theory W 14
Theory X 214
Theory Y 214
TickIT 250–251
time as a resource 153
time and materials contracts 193, 194
time sheets 173
timeboxing in rapid application development 65
timelines 177–179
timing in CPM 122–127
tolerances in PRINCE 2 277
tools 59–61, 71, 84–85, 262
top-down estimating 86
traffic-light method of risk reporting 175
training needs 217
Tuckman, B. W. and Jensen, M. A. on team formation 222

U

UFP, *see* unadjusted function points
unadjusted function points
uncertainty 60–61
 as risk driver 297
 process 61
 product 61
 resource 61
upwards compatibility 243
usability 238
 in ISO 9126 241
 sub-characteristics of 242
usage profile 255
User role in PRINCE 2 271–272

V

- V-process model 66
value for money as a selection criterion 200
value to cost ratios in incremental delivery 75
van Solingen on GQM 257
variance 147
VDM, *see* formal specifications
visualizing progress 175–179
volume tests 203
Vonk, R. on prototyping 68
Vroom on motivation 219–220

W

- warranty period 208
waterfall model 65–66, 299, 310
WBS, *see* work breakdown structure
Weinberg's law 83–84
Weinberg, G. M. on egoless programming 231, 252
Wilemon, D. L. on management tasks 8

Wood on software project management problems 9

work breakdown structure 86, 111–115, 285

activity-based 111–112
hybrid approach 113–115
product-based 112–113

work items 130

Work Packages in PRINCE 2 276

work plan 162

Worseley, C. J., *see* Mayhew, P. J., Worseley, C. J. and Dearnley, P.A.

Y

Youll, David on ball charts 177

Z

Z, *see* formal specifications

Zawacki, R. A., *see* Couger, J. D. and Zawacki, R. A.