3

AN OVERVIEW OF PROJECT PLANNING

OBJECTIVES

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

- approach project planning in an organized step-by-step manner;
- see where the techniques described in other chapters fit into an overall planning approach;
- repeat the planning process in more detail for sets of activities within a project as the time comes to execute them.

3.1 Introduction to Step Wise Project Planning

This chapter describes a framework of basic steps in project planning upon which the following chapters build. Many different techniques can be used in project planning and this chapter gives an overview of the points at which these techniques can be applied during project planning. Chapter 4 will illustrate how different projects may need different technical approaches, but the overall framework should always apply to the planning process.

The framework described is called the Step Wise method to help to distinguish it from other methods such as PRINCE2. PRINCE2 is a set of project management standards that were originally sponsored by what is now the Office of Government Commerce (OGC) for use on British government ICT and business change projects. The standards are now also widely used on non-government projects in the United Kingdom. Step Wise should be compatible with PRINCE2. It should be noted, however, that Step Wise covers only the planning stages of a project and not monitoring and control.

The OGC was previously the CCTA (Central Computing and Telecommunications Agency).

It should be clearly understood that the Step Wise method discussed in this chapter aims to introduce the standardization of the project planning method brought about

Appendix A adds some further details about the PRINCE2 approach.

by PRINCE. PRINCE stands for PRojects IN Controlled Environments. PRINCE2 is in the public do and offers non-proprietary best practice guidance on project management. It is a de facto standard extensively in UK and also internationally. In contrast, the traditional project planning approach discu in many other text books, and practised in many industries allows considerable flexibility in the steps carried out, and the manner in which they are carried out. However, it should be clearly understood th our discussions in the subsequent chapters can be used in a traditional project management situation wi any loss of generality.

In order to illustrate the Step Wise approach and how it might have to be adapted to deal with different cir stances, two parallel examples are used. Let us assume that there are two former Computing and Informa Systems students who now have several years of software development experience under their belts.

CASE STUDY

Example A: A Brightmouth College Payroll

Brigette has been working for the Management Services department of a local authority when she sees an advertisement for the position of Information Systems Development Officer at Brightmouth College. She is attracted to the idea of being her own boss, working in a relatively small organization and helping them to set up appropriate information systems from scratch. She applies for the job and gets it. One of the first tasks that confronts her is the implementation of independent payroll processing. (This scenario has already been used as the basis of some examples in Chapter 1.)

CASE STUDY

Example B: International Office Equipment Annual Maintenance Contracts

Amanda works for International Office Equipment (IOE), which assembles, supplies, instals and services various items of high-technology office equipment. An expanding area of their work is the maintenance of ICT equipment. They have now started to undertake maintenance of equipment of which they were not the original suppliers. An existing application built by the in-house ICT department allows sales staff to input and generate invoices for completed work. A large organization might have to call out IOE several times during a month to deal with problems with equipment. Each month a batch run of the system generates monthly statements for customers so that only one payment a month needs to be made. The management of IOE would like to provide a service where for a single annual payment customers would get free servicing and problem resolution for a pre-specified set of equipment. Amanda has been given her first project management role, the task of implementing this extension to the IOE maintenance jobs billing system.

The enhanced application will need a means of recording the details of the items of equipment to be covered by a customer's annual maintenance contract. The annual fee will depend on the numbers of each type of equipment item that is to be covered. Even though the jobs done under this contract will not be charged for, the work will be recorded to allow for an analysis of costs and the profitability of each customer and each type of equipment. This will provide information which will allow IOE to set future contract prices at an optimally profitable level. At the moment, job details are only recorded after job completion so that invoices can be generated. The new system will allow a central coordinator to allocate jobs to engineers and the system to notify engineers of urgent jobs automatically via their mobile phones.

In Table 3.1 we outline the general approach that might be taken to planning these projects. Figure 3.1 provides an outline of the main planning activities. Steps 1 and 2 'Identify project scope and objectives' and 'Identify project infrastructure' could be tackled in parallel in some cases. Steps 5 and 6 will need to be repeated for each activity in the project.

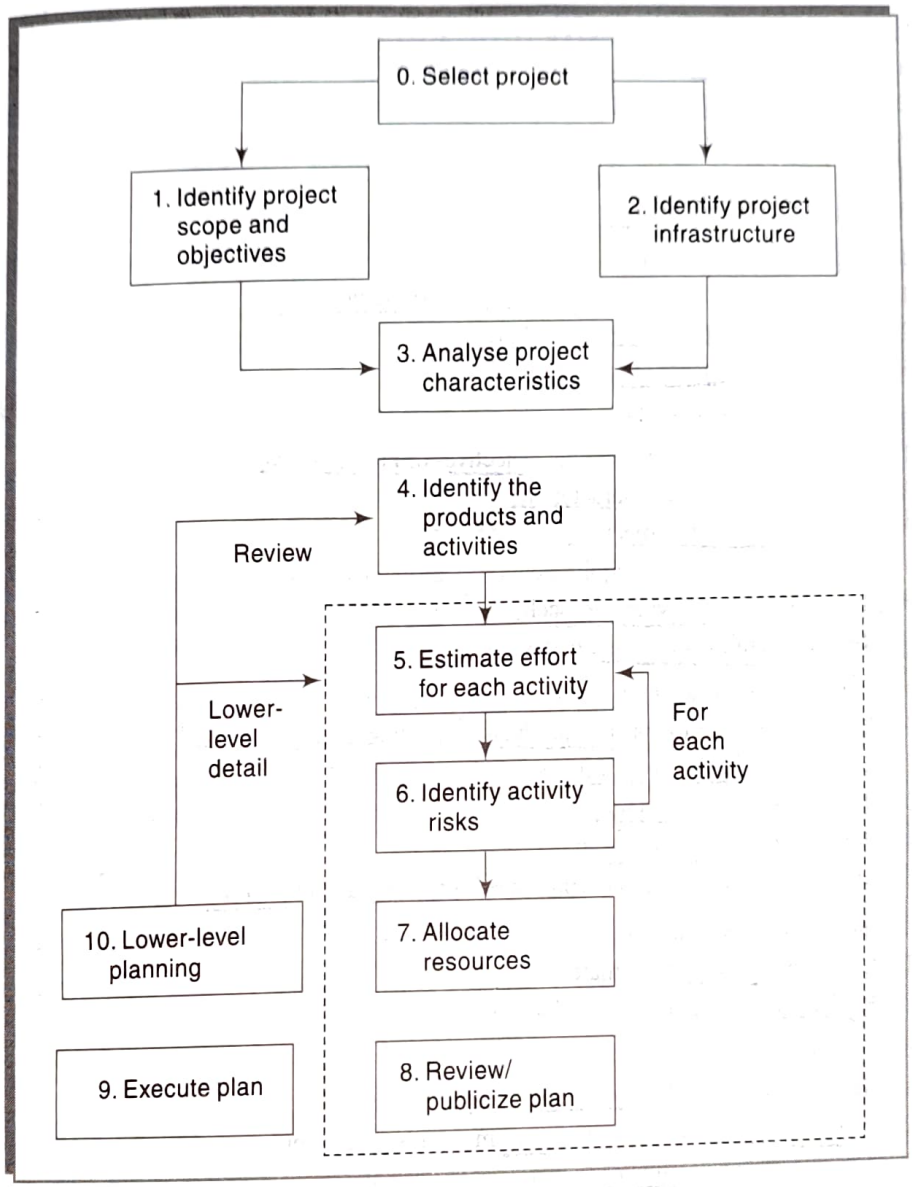


FIGURE 3.1 An overview of Step Wise

A major principle of project planning is to plan in outline first and then in more detail as the time to carry out an activity approaches. Hence the lists of products and activities that are the result of Step 4 will be reviewed when the tasks connected with a particular phase of a project are considered in more detail. This will be followed by a more detailed iteration of Steps 5 to 8 for the phase under consideration.

3.2 Step 0: Select Project

This is called Step 0 because in a way it is outside the main project planning process. Proposed projects do not appear out of thin air – some process must decide to initiate this project rather than some other. While a

TABLE 3.1 An outline of Step Wise planning activities

Step	Activities within step
0	Select project
1	Identify project scope and objectives
	1.1 Identify objectives and measures of effectiveness in meeting them
	1.2 Establish a project authority
	1.3 Identify stakeholders
	1.4 Modify objectives in the light of stakeholder analysis
	1.5 Establish methods of communication with all parties
2	Identify project infrastructure
	2.1 Establish relationship between project and strategic planning
	2.2 Identify installation standards and procedures
	2.3 Identify project team organization
3	Analyse project characteristics
	3.1 Distinguish the project as either objective- or product-driven
	3.2 Analyse other project characteristics
	3.3 Identify high-level project risks
	3.4 Take into account user requirements concerning implementation
	3.5 Select general life-cycle approach
	3.6 Review overall resource estimates
4	Identify project products and activities
	4.1 Identify and describe project products (including quality criteria)
	4.2 Document generic product flows
	4.3 Recognize product instances
	4.4 Produce ideal activity network
	4.5 Modify ideal to take into account need for stages and checkpoints
5	Estimate effort for each activity
	5.1 Carry out bottom-up estimates
	5.2 Revise plan to create controllable activities
6	Identify activity risks
	6.1 Identify and quantify activity-based risks
	6.2 Plan risk reduction and contingency measures where appropriate
	6.3 Adjust plans and estimates to take account of risks
7	Allocate resources
	7.1 Identify and allocate resources
	7.2 Revise plans and estimates to take account of resource constraints
8	Review/publicize plan
	8.1 Review quality aspects of project plan
	8.2 Document plans and obtain agreement
9/10	Execute plan/lower levels of planning
	This may require the reiteration of the planning process at a lower level

feasibility study might suggest that there is a business case for the project, it would still need to be established that it should have priority over other projects. This evaluation of the merits of projects could be part of project portfolio management.

Chapter 2 has already discussed these issues in some detail.

3.3 Step 1: Identify Project Scope and Objectives

The activities in this step ensure that all the parties to the project agree on the objectives and are committed to the success of the project. We have already looked at the importance of the correct definition of objectives in Chapter 1.

Step 1.1: Identify objectives and practical measures of the effectiveness in meeting those objectives

CASE STUDY

Examples: Project Objectives

The project objectives for the Brightmouth College payroll project have already been discussed in Exercise 1.8.

Amanda at JOE has the objectives clearly laid down for her in the recommendations of a business case report which have been accepted by JOE management. The main objectives are to allow:

- details of annual maintenance contracts to be recorded;
- details of maintenance work covered by these contracts to be recorded;
- analysis of costs to be carried out so that the optimal level of maintenance contract fees may be identified;
- recording of job requests and notification of jobs to engineers via mobile phones.

Other objectives are laid down that refer to expected timescales and the resources that might be used.

Step 1.2: Establish a project authority

We have already noted in Chapter 1 that a single overall project authority needs to be established so that there is unity of purpose among all those concerned.

Step 1.3: Stakeholder analysis – identify all stakeholders in the project and their interests

Recall that this was the basis of a discussion in Chapter 1. Essentially all the parties who have an interest in the project need to be identified. In that chapter we listed as an example the stakeholders in the Brightmouth College payroll project.

EXERCISE 3.1

What important stakeholders outside the IOE organization might be considered in the case of the IOE annual maintenance contracts system?

CASE STUDY

Examples: Project Authorities

Throughout the text we use capitalized initial letters to indicate a term that has a precise meaning in the PRINCE2 standards, e.g. Project Board.

Amanda finds that her manager and the main user management have already set up a Project Board which will have overall direction of the project. She is a little concerned as the equipment maintenance staff are organized with different sections dealing with different types of equipment. This means that a customer could have work done by several different sections. Not all the sections are represented on the Project Board and Amanda is aware that there are some differences of opinion between some sections. It is left to the user representatives on the board to resolve those differences and to present an agreed policy to the systems developers.

Brigette finds that effectively she has two different clients for the payroll system: the finance and human resources departments. To help resolve conflicts, it is agreed that the managers of both departments should attend a monthly meeting with the vice-principal which Brigette has arranged in order to steer the project.

Step 1.4: Modify objectives in the light of stakeholder analysis

Compare this with the 'Theory W' of Boehm and Ross mentioned in Chapter 1.

In order to gain the full cooperation of all concerned, it might be necessary to modify the project objectives. This could mean adding new features to the system which are a benefit to some stakeholders as a means of assuring their commitment to the project. This is potentially dangerous as the system size may be increased and the original objectives obscured. Because of these dangers, it is suggested that this process be done consciously and in a controlled manner.

CASE STUDY

Examples: Modified Project Objectives

The IOE maintenance staff are to be given the extra task of entering data about completed jobs. As much customer charges are generated by visits under annual maintenance contracts, engineers may feel that completing cost details is unnecessary bureaucracy, and start to do this in a careless and inaccurate manner. To give some benefit to the engineers, the system is to be extended to reorder spare parts automatically when required. It will also automatically capture timesheet details which previously had to be completed by hand.

At Brightmouth College, the human resources department has a lot of work preparing payroll details for finance. It would be tactful to agree to produce some management information reports for human resources from the payroll details held on the computer.

Step 1.5: Establish methods of communication with all parties

For internal staff this should be fairly straightforward, but a project leader implementing a payroll system would need to find a contact point with BACS (Bankers Automated Clearing Scheme), for instance. This could lead to the first draft of a *communications plan* – to read more about these, see Chapter 12.

3.4 Step 2: Identify Project Infrastructure

Projects are never carried out in a vacuum. There is usually some kind of existing infrastructure into which the project must fit. Where project managers are new to the organization, they must find out the precise nature of this infrastructure. This could be the case where the project manager works for an outside organization carrying out the work for a client.

Step 2.1: Identify relationship between the project and strategic planning

B. Iyer and R. Gottlieb (2004) 'The Four-Domain Architecture: an approach to support enterprise architecture design' *IBM Systems Journal* 43(3) 587–97 provides a good introduction to enterprise architecture concepts.

We saw in Chapter 2 how project portfolio management supported the selection of the projects to be carried out by an organization. Also, how programame management can ensure that a group of projects contribute to a common organizational strategy. There is also a technical framework within which the proposed new systems are to fit. Hardware and software standards, for example, are needed so that various systems can communicate with each other. These technical strategic decisions should be documented as part of an *enterprise architecture* process. Compliance with the enterprise architecture should ensure that successive ICT projects create software and other components compatible with those created by previous projects and also with the existing hardware and software platforms.

CASE STUDY

Examples: Role of Existing Strategic Plans

Enterprise Resource Planning (ERP) systems are integrated software applications usually acquired as off-the-shelf packages that require considerable customization. They integrate all the standard financial and trading applications common to most businesses.

Amanda finds at IOE that there is a well-defined rolling strategic plan which has identified her annual maintenance contracts subsystem as an important required development. Because it is an extension of an existing system, the hardware and software platforms upon which the application are to run are dictated.

Brigette at Brightmouth College finds that there is an overall College strategic plan which describes new courses to be developed, and so on, and mentions in passing the need for 'appropriate administrative procedures' to be in place. There is a recommendation in a consultant's report concerning the implications of financial autonomy that independent payroll processing be implemented as just one module in an ERP system which would cover all the college's financial processing needs. Although the college has quite a lot of ICT equipment for teaching purposes, there is no machine set aside for payroll processing and the intention is that the hardware to run the payroll will be acquired at the same time as the software.

Step 2.2: Identify installation standards and procedures

See discussion of the ISO/IEC 12207 standard in Chapter 1.

Any organization that develops software should define their development procedures. As a minimum, the normal stages in the software life cycle to be carried out should be documented along with the products created at each stage.

Change control and *configuration management* standards should be in place to ensure that changes to requirements are implemented in a safe and orderly way.

See Chapter 9 on monitoring and control.

The procedural standards may lay down the quality checks that need to be done at each point of the project life cycle or these may be documented in a separate *standards and procedures* manual.

The organization, as part of its monitoring and control policy, may have a *measurement programme* in place which dictates that certain statistics have to be collected at various stages of a project.

Finally the project manager should be aware of any *project planning and control standards*. These will specify how the project is controlled; for example, the way that the hours spent by team members on individual tasks are recorded on timesheets.

CASE STUDY

Examples: Identifying Standards

Amanda at IOE finds that there is a very weighty volume of development standards which, among other things, specifies that a specific structured systems analysis and design method be used. She finds that a separate document has been prepared which lays down quality procedures. This specifies when the reviews of work will be carried out and describes detailed procedures governing how the reviews are to be done. Amanda also finds a set of project management guidelines modelled on PRINCE2.

Brigette finds no documents of the nature that Amanda found at IOE except for some handouts for students that have been produced by different lecturers at different times and which seem to contradict each other.

As a stop-gap measure, Brigette writes a brief document which states what the main stages of a 'project' (perhaps 'job for the user' would be a better term in this context) should be. This happens to be very similar to the list given in Chapter 1. She stresses that:

- no job of work to change a system or implement a new one is to be done without there being a detailed specification first;
- the users must record agreement to each specification in writing before the work is carried out.

She draws up a simple procedure for recording all changes to user requirements.

Brigette, of course, has no organizational quality procedures, but she dictates that each person in the group (including herself) has to get someone else to check through their work when they finish a major task and that, before any new or amended software is handed over to the users, someone other than the original developer should test it. She sets up a simple system to record errors found in system testing and their resolution. She also creates a log file of reported user problems with operational systems.

Brigette does not worry about timesheets but arranges an informal meeting with her colleagues each Monday morning to discuss how things are going and also arranges to see the vice-principal, who is her official boss, and the heads of the finance and human resources sections each month to review progress in general terms.

Step 2.3: Identify project team organization

Project leaders, especially in the case of large projects, might have some control over the way that their project team is to be organized. Often, though, the organizational structure will be dictated to them. For example, high-level managerial decision might have been taken that software developers and business analysts will be in different groups, or that the development of business-to-consumer web applications will be done within a separate group from that responsible for 'traditional' database applications.

Some of these issues will be discussed in Chapter 12 on working in teams.

If the project leader does have some control over the project team organization then this would best be considered at a later stage (see Step 7: Allocate resources).

CASE STUDY

Examples: Project Organization

At IOE, there are groups of business analysts set up as teams which deal with individual user departments. Hence the users always know whom they should contact within the information systems department if they have a problem. Software developers, however, work in a 'pool' and are allocated to specific projects on an ad hoc basis.

At Brightmouth College, a software developer has been seconded to Brigette from the technicians supporting the computing courses in the college. She is also allowed to recruit a trainee analyst/programmer. She is not unduly worried about the organizational structure needed.

3.5 Step 3: Analyse Project Characteristics

Chapter 4 elaborates on the process of analysing project characteristics.

The general purpose of this part of the planning operation is to ensure that the appropriate methods are used for the project.

Step 3.1: Distinguish the project as either objective- or product-driven

This has already been discussed in the first chapter. As development of a system advances it tends to become more product-driven, although the underlying objectives always remain and must be respected.

Step 3.2: Analyse other project characteristics (including quality-based ones)

For example, is an information system to be developed or a process control system, or will there be elements of both? Will the system be safety critical, where human life could be threatened by a malfunction?

Step 3.3: Identify high-level project risks

Consideration must be given to the risks that threaten the successful outcome of the project. Generally speaking, most risks can be attributed to the operational or development environment, the technical nature of the project or the type of product being created.

CASE STUDY

Examples: High-Level Risks

We have already noted that Amanda has raised concerns about the possibility that engineers lack the motivation to complete with due care and attention the cost details for jobs done under annual contracts.

Another risk relates to the software functionality which will produce cost analysis reports used for future pricing of annual contracts. If the analysis is incorrect IOE could suffer financially. A decides therefore that the analysis functionality will be produced using an iterative approach, so IOE marketing analyst will look at versions of the reports produced and suggest improvements methods of calculation and presentation before the system is finally made operational.

Brigette at Brightmouth College considers the application area to be very well defined. There is however, that there may be no package on the market that caters for the way that things are done moment. Brigette, therefore, decides that an early task in the project is to obtain information about features of the main payroll packages that are available.

Step 3.4: Take into account user requirements concerning implementation

The clients may have their own procedural requirements. For example, an organization might mandate use of a particular development method.

Step 3.5: Select development methodology and life-cycle approach

The development methodology and project life cycle to be used for the project will be influenced by issues raised above. The idea of a methodology, that is, the group of methods to be used in a project, discussed in Chapter 1. For many software developers, the choice of methods will seem obvious: they use the ones that they have always used in the past. In Chapter 4 we recommend caution in assuming that current project is really similar to previous ones.

Chapter 4 discusses life cycles in more detail.

As well as the methods to be used, there are generic ways of structuring projects, such as the use of the waterfall life cycle outlined in Chapter 4, that need to be considered. While the setting of objectives involves identifying the problems to be solved, part of planning is working out the ways in which these problems are to be solved. In a project that is novel to the planner, some research into the methods typically used in the problem domain worthwhile. For example, sometimes, as part of a project, a questionnaire survey has to be conducted. are lots of books on the techniques used in such surveys and a wise move would be to look at one of them at the planning stage.

Chapter 5 goes into more detail on this topic. Function points are an attempt to measure system size without using lines of code.

Step 3.6: Review overall resource estimates

Once the major risks have been identified and the broad project approach has been decided upon, this would be a good point at which to re-estimate the effort and resources required to implement the project. Where enough information is available, an estimate based on function points might be appropriate.

3.6 Step 4: Identify Project Products and Activities

The more detailed planning of the individual activities now takes place. The longer-term planning is in outline, while the more immediate tasks are planned in some detail.

Step 4.1: Identify and describe project products (or deliverables)

In general, there can be no project products that do not have activities that create them. Wherever possible, we ought also to ensure the reverse: that there are no activities that do not produce a tangible product. Identifying all the things the project is to create helps us to ensure that all the activities we need to carry out are accounted for. Some of these products will be handed over to the client at the end of the project – these are *deliverables*. Other products might not be in the final configuration, but are needed as *intermediate* products used in the process of creating the deliverables.

These products will include a large number of *technical* products, such as training material and operating instructions. There will also be products to do with the *management* and the *quality* of the project. Planning documents would, for example, be management products.

The products will form a hierarchy. The main products will have sets of component products which in turn may have sub-component products, and so on. These relationships can be documented in a Product Breakdown Structure (PBS) – see Figure 3.2. In this example the products have been grouped into those relating to the system as a whole, and those related to individual modules. A third 'group', which happens to have only one product, is called 'management products' and consists of progress reports. The asterisk in the progress reports indicates that there will be new instances of the entity 'progress report' created repeatedly throughout the project.

Note that in Figure 3.2 the only boxes that represent tangible products are those at the bottom of the hierarchy that are not further subdivided. Thus there are only six individual product types shown in the diagram. The boxes that are higher up – for example 'module products' – are simply the names of groups of items.

Some products are created from scratch, for example new software components. A product could quite easily be a document, such as a software design document. It might be a modified version of something that already exists, such as an amended piece of code. A product could even be a person, such as a 'trained user', a product of the process of training. Always remember that a product is the result of an activity. A common error is to identify as products things that are really activities, such as 'training', 'design' and 'testing'. Specifying 'documentation' as a product should also be avoided – by itself this term is just too vague.

PRINCE2 suggests that the PBS be presented as a hierarchy diagram. In practice it may be more convenient to produce a structured list.

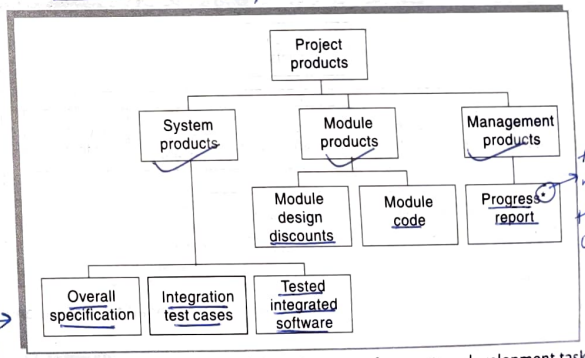


FIGURE 3.2 A fragment of a Product Breakdown Structure for a system development task
* indicates that further progress reports can be added during the course of the project.

This part of the planning process draws heavily on the standards laid down in PRINCE2. These specify that products at the bottom of the PBS should be documented by *Product Descriptions* which contain:

- the name/identity of the product;
- the purpose of the product;
- the derivation of the product (that is, the other products from which it is derived);
- the composition of the product;
- the form of the product;
- the relevant standards;
- the quality criteria that define whether the product is acceptable.

EXERCISE 3.2

At Brightmouth College, Brigette has decided that the finance department at the college should carry out acceptance testing of the new payroll system. This type of testing ensures that the application has been set up in a way that allows the users to carry out their jobs accurately using the new system. As the finance department staff are not sure what test case documents should look like, Brigette draws up a product description of a test case. Write the content for this product description.

CASE STUDY

Examples: Product Breakdown Structures

At IOE, Amanda finds that there is a standard PBS that she can use as a checklist for her own project.

Brigette at Brightmouth College has no installation standard PBS, although she can, of course, refer to various books for standard checklists. She decides that one part of the PBS should contain the products needed to help select the appropriate hardware and software for the payroll application (Figure 3.3).

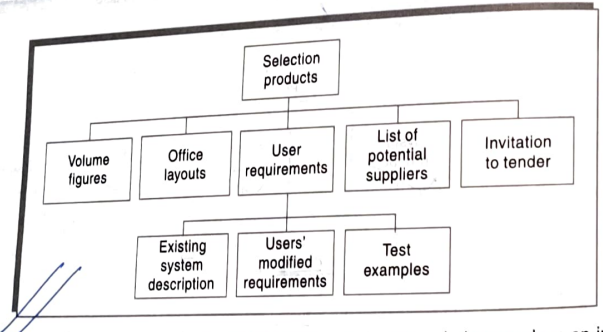


FIGURE 3.3 A Product Breakdown Structure (PBS) for the products needed to produce an invitation to tender (ITT)

EXERCISE 3.3

What would be the product breakdown structure of the deliverables of the vendor who would develop the Brightmouth College payroll software by customizing one of his existing products?

Step 4.2: Document generic product flows

The PFD effectively documents, in outline, the method (see Chapter 1) for the project.

Some products will need one or more other products to exist first before they can be created. For example, a program design must be created before the program can be written and the program specification must exist before the design can be commenced. These relationships can be portrayed in a *Product Flow Diagram* (PFD). Figure 3.4 gives an example. Note that the 'flow' in the diagram is assumed to be from top to bottom and left to right. In the example in Figure 3.4, 'user requirements' is in an oval which means that it is used by the project but is not created by it. It is often convenient to identify an overall product at the bottom of the diagram, in this case 'integrated/tested software', into which all the other products feed.

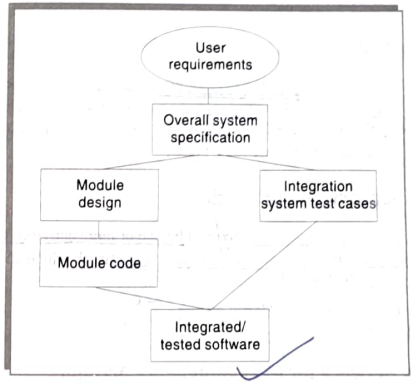


FIGURE 3.4 A fragment of a Product Flow Diagram (PFD) for a software development task

PFDs should not have links between products which loop back iteratively. This is emphatically *not* because iterations are not recognized. On the contrary, the PFD allows for looping back at any point. For example, in the PFD shown in Figure 3.4, say that during integration testing it was found that a user requirement had been missed in the overall system specification. If we go back to overall system specification and change it we can see from the PFD that all the products that follow it might need to be reworked. A new module might need to be designed and coded, test cases would need to be added to check that the new requirements had been successfully incorporated, and the integration testing would need to be repeated.

The form that a PFD takes will depend on assumptions and decisions about how the project is to be carried out. These decisions may not be obvious from the PFD and so a textual description explaining the reasons for the structure can be helpful.

CASE STUDY

Examples: IOE Has Standard PFD

At IOE, Amanda has an installation standard PFD for software development projects. This is because a recognized software development method is used which lays down a sequence of documents that have to be produced. This sequence of products can be straightforwardly documented as a PFD.

EXERCISE 3.4

Draw up a possible Product Flow Diagram (PFD) based on the Product Breakdown Structure (PBS) shown in Figure 3.3. This identifies some of the products of the Brightmouth payroll project, particularly those generated when gathering information to be presented to potential suppliers of the hardware as part of an 'invitation to tender'. The volume figures are such things as the number of employees for whom records will have to be maintained.

Step 4.3: Recognize product instances

This may be delayed to later in the project when more information is known.

Where the same generic PFD fragment relates to more than one instance of a particular type of product, an attempt should be made to identify each of those instances. In the example in Figure 3.2, it could be that in fact there are just two component software modules in the software to be built.

Step 4.4: Produce ideal activity network

In order to generate one product from another there must be one or more activities that carry out the transformation. By identifying these activities we can create an activity network which shows the tasks that have to be carried out and the order in which they have to be executed.

product → Activity → another product

CASE STUDY

Examples: Activity Network for IOE Maintenance Accounts

Part of the initial activity network developed from the PFD in Figure 3.4 for the software development task might look like Figure 3.5.

EXERCISE 3.5

Draw up an activity network for the Product Flow Diagram that you created in Exercise 3.4 (or the PFD given in the solution if you prefer!).

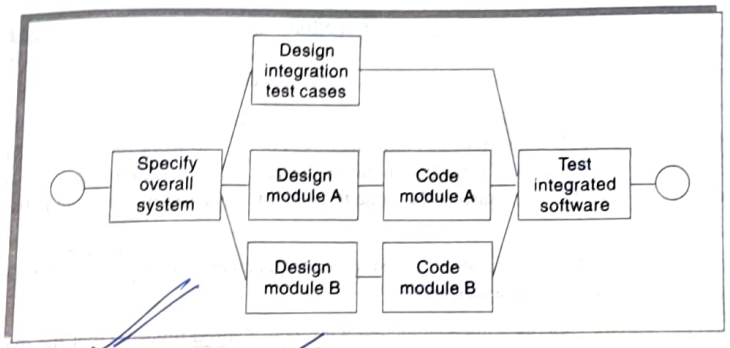


FIGURE 3.5 An example of an activity network

The activity networks are 'ideal' in the sense that no account has been taken of resource constraints. For example, in Figure 3.5, it is assumed that resources are available for both software modules to be developed in parallel. A good rule is that activity networks are never amended to take account of resource constraints.

Step 4.5: Modify the ideal to take into account need for stages and checkpoints

The approach to sequencing activities described above encourages the formulation of a plan which will minimize the overall duration, or 'elapsed time', for the project. It assumes that an activity will start as soon as the preceding ones upon which it depends have been completed.

There might, however, be a need to modify this by dividing the project into stages and introducing checkpoint activities. These are activities which draw together the products of preceding activities to check that they are compatible. This could potentially delay work on some elements of the project – there has to be a trade-off between efficiency and quality.

Strictly, a milestone is a dummy activity with no duration that indicates the start or end of a group of activities. The milestone would therefore be *after* the checkpoint activity.

The people to whom the project manager reports could decide to leave the routine monitoring of activities to the project manager. However, there could be some key activities, or *milestones*, which represent the completion of important stages of the project of which they would want to take particular note. Checkpoint activities are often useful milestones.

EXERCISE 3.6

In the example in Figure 3.5, it has been decided that the designs for modules A and B are to be checked for consistency by 'dry-running' them against the integration test cases before committing staff to software coding. Redraw the activity network to reflect this.

3.7 Step 5: Estimate Effort for Each Activity

Step 5.1: Carry out bottom-up estimates

Some overall estimates of effort, cost and duration will already have been done (see Step 3.6).

Chapter 5 on software effort estimation deals with this topic in more detail.

At this point, estimates of the staff effort required, the probable elapsed time and the non-staff resources needed for each activity will need to be produced. The method of arriving at each of these estimates will vary depending on the type of activity.

The difference between *elapsed time* and *effort* should be noted. Effort is the amount of work that needs to be done. If a task requires three members of staff to work for two full days each, the effort expended is six days. Elapsed time is the time between the start and end of a task. In our example above, if the three members of staff start and finish at the same time then the elapsed time for the activity would be two days.

The individual activity estimates of effort should be summed to get an overall bottom-up estimate which can be reconciled with the previous top-down estimate.

The activities on the activity network can be annotated with their elapsed times so that the overall duration of the project can be calculated.

Step 5.2: Revise plan to create controllable activities

The estimates for individual activities could reveal that some are going to take quite a long time. Long activities make a project difficult to control. If an activity involving system testing is to take 12 weeks, it would be difficult after six weeks to judge accurately whether 50 per cent of the work is completed. It would be better to break this down into a series of smaller subtasks.

CASE STUDY

Examples: IOE Annual Maintenance Contracts – Carry Out Bottom-Up Estimates

At IOE, Amanda has to estimate the lines of code for each of the software modules. She looks at programs that have been coded for similar types of application at IOE in the past to get some idea of the size of the new modules. She then refers to some conversion tables that the information systems development department at IOE have produced which convert the lines of code into estimates of effort. Other tables allow her to allocate the estimated effort to the various stages of the project.

Although Brigette is aware that some additional programs might have to be written to deal with local requirements, the main software is to be obtained 'off the shelf' and so estimating based on lines of code would clearly be inappropriate. Instead, she looks at each individual task and allocates a time. She realizes that in many cases these represent 'targets' as she is uncertain at the moment how long these tasks will really take (see Step 6 below).

There might be a number of activities that are important, but individually take up very little time. For a training course, there might be a need to book rooms and equipment, notify those attending, register students on the training system, order refreshments, copy training materials, and so on. In a situation like this it would be easier to bundle the activities into a single merged activity 'make training course arrangements' which could be supplemented with a checklist.

In general, try to make activities about the length of the reporting period used for monitoring and controlling the project. If you have a progress meeting every two weeks, then it would convenient to have activities of two weeks' duration on average, so that progress meetings would normally be made aware of completed tasks each time they are held.

3.8 Step 6: Identify Activity Risks

Step 6.1: Identify and quantify activity-based risks

Chapter 7 on risk touches on this topic in more detail.

Risks inherent in the overall nature of the project have already been considered in Step 3. We now want to look at each activity in turn and assess the risks to its successful outcome. Any plan is always based on certain assumptions. Say the design of a component is planned to take five days. This is based on the assumption that the client's requirement is clear and unambiguous. If it is not then additional effort to clarify the requirement would be needed. The possibility that an assumption upon which a plan is based is incorrect constitutes a risk. In this example, one way of expressing the uncertainty would be to express the estimate of effort as a range of values.

As will be seen in Chapter 7, a simple way of dealing with uncertainty is to have a 'most likely' estimate for where everything works with no problems (such as users changing their requirements) and a second estimate that includes a safety margin so that it has an estimated 95 per cent chance of being met.

A project plan will be based on a huge number of assumptions, and so some way of picking out the risks that are most important is needed. The damage that each risk could cause and the likelihood of it occurring have to be gauged. This assessment can draw attention to the most serious risks. The usual effect if a problem materializes is to make the task longer or more costly.

Step 6.2: Plan risk reduction and contingency measures where appropriate

It may be possible to avoid or at least reduce some of the identified risks. On the other hand, *contingency plans* specify action that is to be taken if a risk materializes. For example, a contingency plan could be to use contract staff if a member of the project team is unavailable at a key time because of serious illness.

Step 6.3: Adjust overall plans and estimates to take account of risks

We may change our plans, perhaps by adding new activities which reduce risks. For example, a new programming language might mean we schedule training courses and time for the programmers to practise their new programming skills on some non-essential work.

CASE STUDY

Examples: Identifying Risks

As well as the new software modules that will have to be written, Amanda has identified several existing modules that will need to be amended. The ease with which the modules can be amended will depend upon the way that they were originally written. There is therefore a risk that they may take longer than expected to modify.

Amanda takes no risk reduction measures as such but notes a pessimistic elapsed time for the amendment activity.

Brigette identifies as a risk the possible absence of key staff when investigating the user requirements, as this activity will take place over the holiday period. To reduce this risk, she adds a new activity, 'arrange user interviews', at the beginning of the project. This will give her advance notice of any likely problems of this nature.

3.9 Step 7: Allocate Resources

Step 7.1: Identify and allocate resources

Chapter 8 on resource allocation covers this topic in more detail.

The type of staff needed for each activity is recorded. The staff available for the project are identified and are provisionally allocated to tasks.

Step 7.2: Revise plans and estimates to take into account resource constraints

Some staff may be needed for more than one task at the same time and, in this case, an order of priority is established. The decisions made here may have an effect on the overall duration of the project when some tasks are delayed while waiting for staff to become free.

Ensuring someone is available to start work on an activity as soon as the preceding activities have been completed might mean that they are idle while waiting for the job to start and are therefore used inefficiently.

Gantt charts are named after Henry Gantt and 'Gantt' should therefore not be written in capital letters as if it stood for something!

The product of Steps 7.1 and 7.2 would typically be a Gantt chart – see Figure 3.6. The Gantt chart gives a clear picture of when activities will actually take place and highlights which ones will be executed at the same time. Activity networks can be misleading in this respect.

CASE STUDY

Examples: Taking Resource Constraints into Account

Amanda has now identified three new major software modules plus an existing software module that will need extensive amendment. At IOE the specification of modules is carried out by the lead systems analyst for the project (who in this case is Amanda) assisted by junior analyst/designers. Four analyst/programmers are available to carry out the design, coding and unit testing of the individual modules. After careful consideration and discussion with her manager, Amanda decides to use only three analyst/programmers so as to minimize the risk of staff waiting between tasks and thus reduce staff costs. It is accepted that this decision, while reducing the cost of the project, will delay its end.

Brigette finds that she herself will have to carry out many important activities. She can reduce the workload on herself by delegating some work to her two colleagues, but she realizes that she will have to devote more time to specifying exactly what they will have to do and to checking their work. She adjusts her plan accordingly.

3.10 Step 8: Review/Publicize Plan

Step 8.1: Review quality aspects of the project plan

A danger when controlling any project is that an activity can reveal that an earlier activity was not properly completed and needs to be reworked. This, at a stroke, can transform a project that appears to be progressing satisfactorily into one that is badly out of control. It is important to know that when a task is reported as completed, it really is – hence the importance of quality reviews. Each task should have *quality criteria*. These are quality checks that have to be passed before the activity can be 'signed off' as completed.

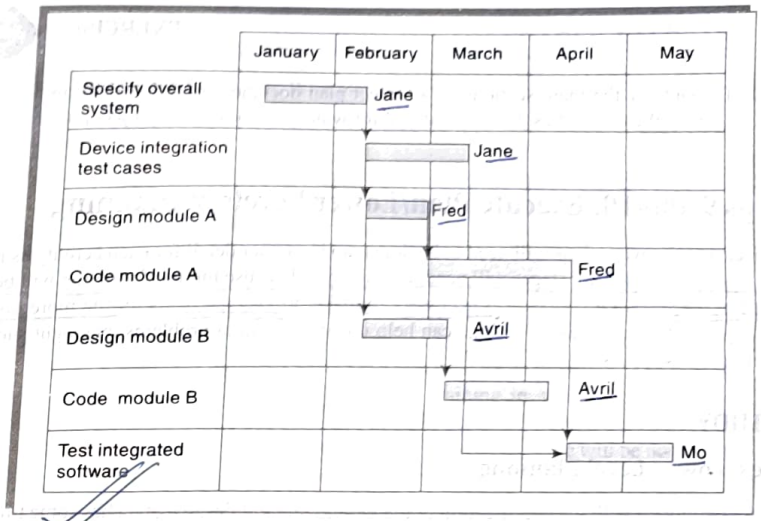


FIGURE 3.6 Gantt chart showing when staff will be carrying out tasks

CASE STUDY

Examples: IOE Existing Quality Standards

Amanda finds that at IOE, the Quality Standards and Procedures Manual lays down quality criteria for each type of task. For example, all module design documentation for any group of modules that interact with one another has to be reviewed by a group of colleagues before the coding can commence. This is to reduce the likelihood of integration problems when the components are finally executed together. Amanda adds an activity to her plan to deal with this.

EXERCISE 3.7

Brigette has no installation standards to help her apart from the minimal ones she has written herself. What quality checks might Brigette introduce to ensure that she has understood the users' requirements properly?

Step 8.2: Document plans and obtain agreement

It is important that the plans be carefully documented and that all the parties to the project understand and agree to the commitments required of them in the plan. This may sound obvious, but it is amazing how often this is not done. Chapter 12 describes the use of a *communications plan* to ensure appropriate communications between stakeholders at the right points in the project.

EXERCISE 3.8

At the end of Chapter 1 the main sections of a project plan document were listed. Draw up a table showing which Step Wise activities provide material for which sections of the project plan.

3.11 Steps 9 and 10: Execute Plan/Lower Levels of Planning

Once the project is under way, plans will need to be drawn up in greater detail for each activity as it becomes due. Detailed planning of the later stages will need to be delayed because more information will be available nearer the start of the stage. Of course, it is necessary to make provisional plans for the more distant tasks because thinking about what needs to be done can help unearth potential problems, but sight should not be lost of the fact that these plans are provisional.

CASE STUDY

Examples: Lower-Level Planning

While work is going on with the specification of the individual modules, Amanda has some time to start planning the integration tests in some detail. She finds that one of the modules – the one that deals with recording job requests – does not actually communicate directly with the other new modules and can therefore be reviewed independently of the others. She schedules an earlier review of this module as this allows coding of the module to be started earlier.

When Brigette comes to consider the activity 'draft invitation to tender', she has to familiarize herself with the detailed institutional rules and procedures that govern this process. She finds that in order to draft this document she will need to obtain some additional pieces of information from the users.

CONCLUSION

This chapter has presented a framework into which the techniques described in the other parts of the book should slot. It is suggested that any planning approach should have the following elements:

- the establishment of project objectives;
- the analysis of the characteristics of the project;
- the establishment of an infrastructure consisting of an appropriate organization and set of standard methods and tools;
- the identification of the products of the project and the activities needed to generate those products;
- the allocation of resources to activities;
- the establishment of quality controls.

Project planning is an iterative process. As the time approaches for particular activities to be carried out they should be replanned in more detail.

FURTHER EXERCISES

1. List the products created by the Step Wise planning process.
2. What products must exist before the activity 'test program' can take place? What products does this activity create?
3. An employee of a training organization has the task of creating case study exercises and solutions for a training course which teaches a new systems analysis and design method. The person's work plan has a three-week task 'learn new method'. A colleague suggests that this is unsatisfactory as a task as there are no concrete deliverables or products from the activity. What can be done about this?
4. In order to carry out usability tests for a new word processing package, the software has to be written and debugged. User instructions have to be available describing how the package is to be used. These have to be scrutinized in order to plan and design the tests. Subjects who will use the package in the tests will need to be selected. As part of this selection process, they will have to complete a questionnaire giving details of their past experience of, and training in, typing and using word processing packages. The subjects will carry out the required tasks using the word processing package. The tasks will be timed and any problems the subjects encounter with the package will be noted. After the test, the subjects will complete another questionnaire about what they felt about the package. All the data from the tests will be analysed and a report containing recommendations for changes to the package will be drawn up. Draw up a Product Breakdown Structure, a Product Flow Diagram and a preliminary activity network for the above.
5. Question 4 in the further exercises for Chapter 1 refers to a scenario relating to a training exercise. Using that scenario, draw up a Product Breakdown Structure, a Product Flow Diagram and a preliminary activity network.
6. Brightmouth College intends to automate the routine activities of its library including issuing books to users, return of books, handling fine collection, and querying availability of books. The library has around 10,000 books. At present, the activities of the library are being carried out manually by the four member library staff. The college intends to allot the development of the software to a vendor. The software would have to be transferred to the library in a fully operational mode. To speed up the delivery of software, the vendor would have to create the operational database during the development of the software. This would involve entering details of the existing books into a CSV (comma separated values) file. After the development of the software, the CSV data will have to be imported into the software. After alpha testing, the software would have to be tested in the operational environment. For this, the software would have to be run along side the manual system at the library for a week. During this time, user training would also have to be conducted.
   (a) Identify and represent the deliverables using a product breakdown structure (PBS)
   (b) Develop the product flow diagram
   (c) Develop an activity network

4

SELECTION OF AN APPROPRIATE PROJECT APPROACH

OBJECTIVES

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

- evaluate situations where software applications could be acquired off-the-shelf rather than being built specially;
- 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 some risks by the creation of appropriate prototypes;
- reduce other risks by implementing the project in increments;
- identify where unnecessary organizational obstacles can be removed by using agile development methods.

4.1 Introduction

The development of software in-house usually means that:

- the developers and the users belong to the same organization;
- the application will slot into a portfolio of existing computer-based systems;
- the methodologies and technologies are largely dictated by organizational standards and policies including the existing enterprise architecture.

However, a software supplier could carry out successive development projects for a variety of external customers. They would need to review the methodologies and technologies to be used for each individual project. This decision-making process has been called *technical planning* by some, although here we use the term *project analysis*. Other terms for this process are *methods engineering* and *methods tailoring*. Even when

development is in-house, any characteristics of the new project requiring a different approach from previous projects need to be considered. A wide range of system development methods exists, but many organizations get along without using any of the recognized approaches. Where methods are used, 'means–end inversion' can happen: developers focus on the means – the procedures and intermediate products of a prescribed method – at the expense of the 'end', the actual required outcomes of the work. These issues are the subject of this chapter.

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

B. Fitzgerald, N. L. Russo and T. O'Kane (2003). 'Software development method tailoring at Motorola' *Communications of the ACM* 46(4) 65–70 provides a good insight into how method tailoring works in practice.

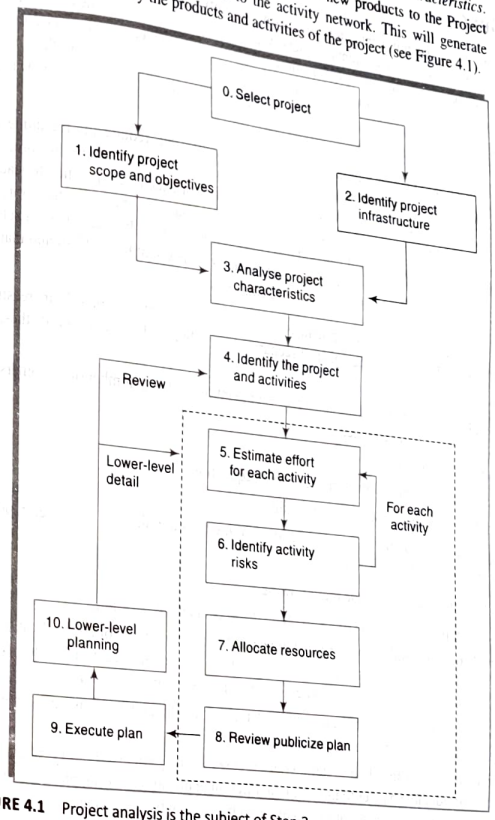


FIGURE 4.1 Project analysis is the subject of Step 3

In the remainder of this chapter we will look at how the characteristics of a project's environment and the application to be delivered influence the shape of the plan of a project. We will then look at some of the more common *process models*, namely the waterfall approach, prototyping and incremental delivery. Some of the ideas of prototyping and incremental delivery have been further developed and made part of *agile methods*. We will have a look at how these *lightweight* processes have been designed to remove what have been seen as the bureaucratic obstacles created by more formal, *heavyweight* methods.

4.2 Build or Buy?

The communication challenges of geographically dispersed projects are discussed in Chapter 12 on working in teams.

Software development can be seen from two differing viewpoints: that of the developers and that of the clients or users. With *in-house development*, the developers and the users are in the same organization. Where the development is *outsourced*, they are in different organizations. In these days of global system development, the different organizations could be on different continents. These factors will affect the way that a project is organized.

The development of a new IT application within an organization would often require the recruitment of technical staff who, once the project has been completed, will no longer be required. Because this project is a unique new development for the client organization there may be a lack of executives qualified to lead the effort. Contracting the project out to an external IT development company may be attractive in these circumstances. The contracting company will have technical and project expertise not readily available to the client. However, there would still be considerable management effort needed by the client to establish and manage the contract and this is the subject of Chapter 10 on managing contracts.

Whether in-house or outsourced, software development is still involved. An option which is increasingly taken – as in the case of the Brightmouth College payroll scenario – is to obtain a licence to run off-the-shelf software. The advantages of such an approach include:

- the supplier of the application can spread the cost of development over a large number of customers and thus the cost per customer should be reduced;
- the software already exists and so
  - it can be examined and perhaps even trialed before acquisition,
  - there is no delay while the software is being built;
- where lots of people have already used the software, most of the bugs are likely to have been reported and removed, leading to more reliable software.

However, there are disadvantages, which include the following.

- As you have the same application as everyone else, there is no competitive advantage.
- Modern off-the-shelf software tends to be very customizable: the characteristics of the application can be changed by means of various parameter tables. However, this flexibility has limits and you may end up having to change your office procedures in order to fit in with the computer system.
- You will not own the software code. This may rule out making modifications to the application in response to changes in the organization or its environment.
- Once you have acquired your off-the-shelf system, your organization may come to be very reliant upon it. This may create a considerable barrier to moving to a different application. The supplier may be in a position to charge inflated licence fees because you are effectively a captive customer.

We will explore these issues further in Chapter 10 on managing contracts. In the remainder of this chapter we focus on situations where new software is being developed, whether in-house or outsourced.

4.3 Choosing Methodologies and Technologies

Strictly speaking, methodology should refer to the 'study of methods'.

In the context of ICT system development and software engineering, the term *methodology* describes a collection of *methods*. We introduced 'method' in Chapter 1 as a general way of carrying out a specific task that could be applicable to any project needing to do that task. *Techniques* and methods are sometimes distinguished. Techniques tend to involve the application of scientific, mathematical or logical principles to resolve a particular kind of problem. They often require the practice of particular personal skills (the word 'technique' is derived from the Greek for skilful) – software design is a good example. Methods often involve the creation of *models*. A model is a representation of a system which abstracts certain features but ignores others. For example, an entity relationship diagram (ERD) is a model of the structure of the data used by a system. What can be confusing is that a software development life cycle itself is a type of system. Features of life cycles can therefore be abstracted and represented as 'models' as we will see later in this chapter. Some of these models can thus start to look a bit like methods.

Project analysis should select the most appropriate methodologies and technologies for a project. Methodologies include approaches like the Unified Software Development Process (USDP), Structured Systems Analysis and Design Method (SSADM) and Human-Centred Design, while technologies might include appropriate application-building and automated testing environments. The analysis identifies the methodology, but also selects the methods within the methodology that are to be deployed.

As well as the products and activities, the chosen methods and technologies will affect:

- 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 objective-driven or product-driven

The soft systems approach is described in P. Checkland and J. Scholes (1999) *Soft Systems Methodology in Action*, John Wiley and Sons.

In Chapter 1 we distinguished between *objective-driven* and *product-driven* projects. A product-driven project creates products defined before the start of the project. An objective-driven project will often have come first which will have defined the general software solution that is to be implemented.

The project manager's dream is to have well-defined objectives but as much freedom as possible about how those objectives are to be satisfied. An objective might be to pay staff in a start-up company reliably, accurately and with low administrative costs. The company does not have to specify the use of a particular packaged software solution at the outset – but as we will see, there can be exceptions to this.

Sometimes the objectives of the project are uncertain or are the subject of disagreement. People might be experiencing problems but no one knows exactly how to solve these problems. ICT specialists might provide

help with some problems but assistance from other specialisms might be needed with others. In these kinds of situation a *soft systems* approach might be considered.

Analyse other project characteristics

The following questions can be usefully asked.

- *Is a data-oriented or process-oriented system to be implemented?* Data-oriented systems generally mean information systems that will have a substantial database. Process-oriented systems refer to embedded control systems. It is not uncommon to have systems with elements of both. Some writers suggest that the OO approach is more suitable for process-oriented systems where control is important than for systems dominated by a relational database.

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

- *Will the software that is to be produced be a general tool or application specific?* An example of a general tool would be a spreadsheet or a word processing package. An application-specific package could be, for example, an airline seat reservation system.
- *Are there specific tools available for implementing the particular type of application?* For example:
  - *does it involve concurrent processing?* – 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 rules which result in some 'expert advice' when applied to a problem, and specific methods and tools exist for developing such systems; or
  - *will the system to be produced make heavy use of computer graphics?*

> Note that here we are talking about writing the software tool, not its use.

- *Is the system to be created safety critical?* For instance, could a malfunction in the system endanger human life? If so, among other things, testing would become very important.
- *Is the system designed primarily to carry out predefined services or to be engaging and entertaining?* With software designed for entertainment, design and evaluation will need to be carried out differently from more conventional software products.
- *What is the nature of the hardware/software environment in which the system will operate?* The environment in which the final software will operate could be different from that in which it is to be developed. Embedded software might be developed on a large development machine which has lots of supporting software tools such as compilers, debuggers and static analysers, but then be downloaded to a small processor in the target configuration. A standalone desktop application needs a different approach to one for a mainframe 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 which holds details of the plans of plant used by a water company to supply water to consumers;

(d) a software package to support project managers;

(e) a system used by lawyers to access case law relating to company taxation.

Identify high-level project risks

At the beginning of a project, some managers might expect elaborate plans even though we are ignorant of many important factors affecting the project. For example, until we have analysed the users' requirements in detail we cannot estimate the effort needed to build a system to meet those requirements. The greater the uncertainties at the beginning, the greater the risk that the project will be unsuccessful. Once we recognize an area of uncertainty we can, however, take steps to reduce its uncertainty.

> Chapter 2 has already touched on some aspects of risk which are developed further in Chapter 7.

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

- *Product uncertainty* How well are the requirements understood? The users themselves could be uncertain about what a proposed information system is to do. The government, say, might introduce a new form of taxation but its detailed operation might not be known until case law has been built up. Some environments change so quickly that a seemingly precise and valid statement of requirements rapidly becomes out of date.
- *Process uncertainty* The project under consideration might be the first where an organization is using an approach like extreme programming (XP) or a new application-building tool. Any change in the way that the systems are developed introduces uncertainty.

> Extreme programming will be discussed in Section 4.15.

- *Resource uncertainty* The main area of uncertainty here is likely to be the availability of staff of the right ability and experience. The larger the number of resources needed or the longer the duration of the project, the more inherently risky it will be.

> Of course, some risk factors can increase both uncertainty and complexity.

Some factors – such as continually changing requirements – increase *uncertainty*, while others – for instance, software size – increase *complexity*. Different strategies are needed to deal with the two distinct types of risks.

EXERCISE 4.2

At IOE, Amanda has identified possible user resistance as a risk to the annual maintenance contracts project. Would you classify this as a product, process or resource risk? It may be that it does not fit into any of these categories and some other is needed.

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

Take into account user requirements concerning implementation

We suggested earlier that staff planning a project should try to ensure that unnecessary constraints are not imposed on the way that a project's objectives are to be met. The example given was the specification of the exact payroll package to be deployed. Sometimes, such constraints are unavoidable. International conglomerates have found that imposing uniform applications and technologies throughout all their component parts can save time and money. Obtaining IT services for the whole organization from a single supplier can mean that large discounts can be negotiated.

> Chapter 13 on software quality discusses BS EN ISO 9001.

A client organization often lays down standards that have to be adopted by any contractor providing software for them. Sometimes organizations specify that suppliers of software have BS EN ISO 9001 : 2000 or TickIT accreditation. This will affect the way projects are conducted.

Select general life-cycle approach

- *Control systems* A real-time system will need to be implemented using an appropriate methodology. Real-time systems that employ concurrent processing may have to 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 would be especially appropriate where the project employs a large number of development staff whose work will need to be coordinated: the method lays down in detail the activities and products needed at each step. Team members would therefore know exactly what is expected.

SSADM as a named methodology is now rarely used, but many of methods within it are still in wide use – sometimes under the general name of business system development (BSD) techniques.

- *Availability of users* Where the software is for the general market rather than application and user specific, then a methodology which assumes that identifiable users exist who can be quizzed about their needs would have to be thought about with caution. Some business systems development methods assume an existing clerical system which can be analysed to yield the logical features of a new, computer-based, system. In these cases a marketing specialist may act as a surrogate user.
- *Specialized techniques* For example, expert system shells and logic-based programming languages have been invented to expedite the development of *knowledge-based systems*. Similarly, a number of specialized techniques and standard components are available to assist in the development of *graphics-based systems*.
- *Hardware environment* The environment in which the system is to operate could put constraints on the way it is to be implemented. The need for a fast response time or restricted computer memory might mean that only low-level programming languages can be used.
- *Safety-critical systems* Where safety and reliability are essential, this might justify the additional expense of a formal specification using a notation such as OCL. Extremely critical systems could justify the cost of having independent teams develop parallel systems with the same functionality. The operational systems can then run concurrently with continuous cross-checking. This is known as *n-version programming*.

OCL stands for Object Constraint Language.

- *Imprecise requirements* Uncertainties or a *novel hardware/software platform* 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

What, in broad outline, would be the most suitable approach for each of the following?

(a) a system which 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 in the Channel Tunnel.

4.4 Software Processes and Process Models

A software product development process usually starts when a request for the product is received from the customer. For a generic product, the marketing department of the company is usually considered as the customer. This expression of need for the product is called product inception. From the inception stage, a product undergoes a series of transformations through a few identifiable stages until it is fully developed and released to the customer. After release, the product is used by the customer and during this time the product needs to be maintained for fixing bugs and enhancing functionalities. This stage is called the maintenance stage. When the product is no longer useful to the customer, it is retired. This set of identifiable stages through which a product transits from inception to retirement form the life cycle of the product. The software life cycle is also commonly referred to as Software Development Life Cycle (SDLC) and software process.

A life cycle model (also called a process model) of a software product is a graphical or textual representation of its life cycle. Additionally, a process model may describe the details of various types of activities carried out during the different phases and the documents produced.

4.5 Choice of Process Models

The word 'process' emphasizes 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 applies to the development of computer-based applications. 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*.

The planner selects methods and specifies how they are to be applied. Not all parts of a methodology such as USDP or SSADM will be 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 explicitly stated.

4.6 Structure versus Speed of Delivery

The principle behind structured methods is 'get it right first time'.

Although some 'object-oriented' specialists might object(!), we include the OO approach as a structured method – after all, we hope it is not unstructured. Structured methods consist of sets of steps and rules which, when applied, generate system products such as use case diagrams. Each of these products is carefully defined. Such methods are more time consuming and expensive than more intuitive approaches. The pay-off, it is hoped, is 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. Because of the additional effort needed their greater applicability to large and complex projects, these are often called *heavyweight* methods.

It might be thought that users would generally welcome the more professional approach that struc methods imply. However, customers for software are concerned with getting working applications deli quickly and at less cost and often see structured methods as unnecessarily bureaucratic and slow. One respo to this has been *rapid application development* (RAD) which puts the emphasis on quickly producing p types of the software for users to evaluate.

Joint Application Development by Jane Wood and Denise Silver, Wiley and Sons, 1995, is a useful introduction to JAD.

The RAD approach does not preclude the use of some elements of structured meth such as the drafting of logical data structure diagrams but also adopts tactics as *joint application development* (JAD) workshops. In these workshops, develo and users work together intensively for, say, three to five days and identify and fully documented system requirements. Often these workshops are conducted a from the normal business and development environments in *clean rooms*, spe conference rooms free from outside interruption and suitably furnished with whiteboards and other aid communication. Advocates of JAD believe that these hot-house conditions can speed up communication negotiation that might otherwise take several weeks or months.

Use of JAD does not mean that the project is not structured. The definition of the scope of the project initial research involving the interviewing of key personnel and the creation of preliminary data and pro models would need to planned and executed before the JAD sessions were organized. The results of sessions could be implemented using quite conventional methods.

Another way of speeding up delivery is simply to deliver less. This can be done by breaking a large d opment into a series of small increments, each of which delivers a small amount of useful functio quickly.

Two competing pressures can be seen. One is to get the job done as quickly and cheaply as possible, an other is to make sure that the final product has a robust structure which will be able to meet evolving r Later in this chapter and in Chapter 12 we will discuss the increasingly important topic of agile met which focuses on lightweight processes. There is, however, a contrasting approach which is the attem create *model-driven architectures* (MDA). System development using MDA involves creating a plat independent model (PIM) which specifies system functionality using UML diagrams supplemente additional information recorded in the Object Constraint Language (OCL). A PIM is the logical structu should apply regardless of the software and hardware environment in which the system is to be implem This can be transformed into a platform-specific model (PSM) that takes account of a particular develop and implementation environment. A PSM can then be transformed into executable code to implem working system. The goal is that once a PIM had been created the creation of PSMs and executable cod be automated. At present, the automation of these transformation processes is still being developed.

4.7 The Waterfall Model

This is the 'classical' model of system development that is also known as the *one-shot* or *once-through* m As can be seen from the example in Figure 4.2, there is a sequence of activities working from top to bo The diagram shows some arrows pointing upwards and backwards. This indicates that a later stage reveal the need for some extra work at an earlier stage, but this should definitely be the exception rathe the rule. After all, the flow of a waterfall should be downwards, with the possibility of just a little spil

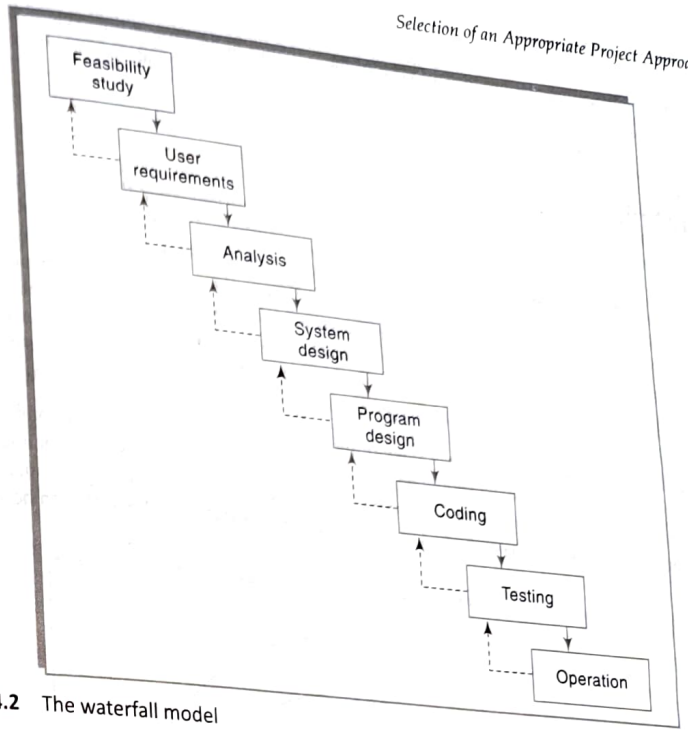


FIGURE 4.2 The waterfall model

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 reworking tasks previously thought to be completed. Having to reopen completed activities 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).

The waterfall approach may be favoured by some managements because it creates natural milestones at the end of each phase. At these points, managers can review project progress to see whether the business case for the project is still valid. This is sometimes referred to as the stage-gate model. As we will see, stage-gates are compatible with process models other than the waterfall, but higher management may have to accept that activities may have to be grouped in different ways with these alternative approaches.

Even though writers often advocate alternative models, there is nothing intrinsically wrong with the waterfall approach in the right place. It is the ideal for which the project manager strives. Where the requirements are well defined and the development methods are well understood, the waterfall approach allows project completion times to be forecast with some confidence, allowing the effective control of the project. However, where there is uncertainty about how a system is to be implemented, and unfortunately there very often is, a more flexible, iterative, approach is required.

The waterfall model can expanded into the **V-process model** which is further explored in Section 13.11 on testing. This expansion is done by expanding the testing process into different types of testing which check

the executable code against the products of each of the activities in the project life cycle leading up to coding. For example, the code may seem to execute correctly, but may be at variance with the system design. This is explained further in Chapter 13.

4.8 The Spiral Model

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).

It could be argued that this is another way of looking at the waterfall model. At the end of each stage in the waterfall model, it is possible to escape. At the end of any activity in the sequence, a feasibility study might decide that the implementation of a proposed system would not be beneficial. The management therefore authorize work on the detailed specification of user requirements. Some analysis, for instance the interviewing of users, will already have taken place at the feasibility stage, but a more thorough investigation is now launched. This could reveal that the costs of implementing the system would be higher than projected benefits and lead to a decision to abandon the project.

A greater level of detail is considered at each stage of the project and a greater degree of confidence in the probability of success for the project should be justified. This can be portrayed as a loop or a spiral. The system to be implemented is considered in more detail in each sweep. Each sweep terminates with evaluation before the next iteration is embarked upon. Figure 4.3 illustrates how SSADM can be interpreted in such a way.

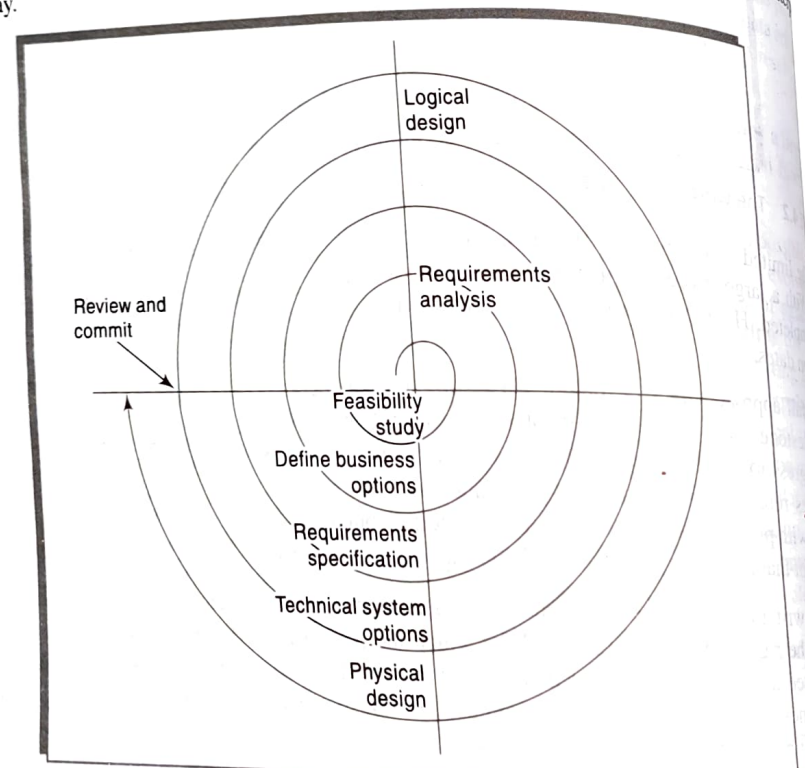


FIGURE 4.3 The application of the spiral model to SSADM version 4

A key point here is that uncertainty about a project is usually because of a lack of knowledge about some aspect. We can spend money on activities at the start of the project that buy knowledge and reduce that uncertainty.

The distinguishing characteristic features of the spiral model are the incremental style of development and the ability to handle various types of risks. Each loop of the spiral is called a phase of this software process. In each phase, one or more features of the product are implemented after resolving any associated risks through prototyping. The exact number of loops of the spiral is not fixed and varies from one project to another. Note that the number of loops shown in Figure 4.4 is just an example illustrating how the spiral model can subsume SSADM. Each loop of the spiral is divided into four quadrants, indicating four stages in each phase. In the first stage of a phase, one or more features of the product are analysed and the risks in implementing those features are identified and resolved through prototyping. In the third stage, the identified features are implemented using the waterfall model. In the fourth and final stage, the developed increment is reviewed by the customer along with the development team and the features to be implemented next are identified. Note that the spiral model provides much more flexibility compared to the other models, in the sense that the exact number of phases through which the product is to be developed can be tailored by the project manager during execution of the project.

4.9 Software Prototyping

This is one way in which we can buy knowledge and reduce uncertainty. 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 or evolutionary.

- *Throw-away prototypes* The prototype tests out some ideas and is then discarded when the true development of the operational system is commenced. The prototype could be developed using a different software or hardware environment. For example, a desktop application builder could be used to evolve an acceptable user interface. A procedural programming language is then used for the final system where machine-efficiency is important.
- *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.

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

- *Learning by doing* We can usually look back on a task and see where we have made mistakes.
- *Improved communication* Users do not get a feel for how the system is likely to work in practice from a specification.
- *Improved user involvement* The users can be more actively involved in design decisions.
- *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 by trying out prototypes.
- *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. The humble spreadsheet can, for instance, check that calculations have been specified correctly.

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

Some may argue, however, that this is a very dangerous suggestion.

- *Reduced need for documentation* Because a working prototype can be examined there is less need for detailed documentation of requirements.
- *Reduced maintenance costs* If the user is unable to suggest modifications at the prototyping stage they are more likely to ask for changes to the operational system. This reduction of maintenance costs is the core of the financial case for 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 cases is generally not the creation of the test input but the accurate calculation of the expected results. A prototype can help here.

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

- *Users can misunderstand the role of the prototype* For example, they might expect the prototype to have as stringent input validation or as fast a response as the operational system, although this was not intended.
- *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 can be difficult to control the prototyping cycle if the driving force is the user's propensity to try out new things.
- *Additional expense* Building and exercising a prototype will incur additional expenses. However, this should not be over-estimated as many analysis and design tasks have to be undertaken whatever the approach.
- *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 could mean that code developers have to be sited close to the users. One trend is for organizations in developed countries to transfer software development to developing countries with lower costs such as India. Prototyping might prevent this.

4.10 Other Ways of Categorizing Prototypes

What is being learnt?

The most important reason for prototyping is a need to learn about an area of uncertainty. Thus it is essential to identify at the outset what is to be learnt from the prototype.

Computing students often realize that the software that they are to write as part of their final-year 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 can be used to find out about new development techniques, by using them in a pilot project. Alternatively, the development methods might be well known, but the nature of the application 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 usually simulates only some aspects of the target application. For example there might be:

- *Mock-ups* As when copies of input screens 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 show 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, but not all, features are prototyped fully.
  - *Horizontal* All features are prototyped but not in detail – perhaps there is not full validation of input.

What is being prototyped?

- *The human–computer interface* With business applications, business process requirements have usually been established at an early stage. Prototyping tends, therefore, to be confined to the nature of operator interaction. Here the physical vehicle for the prototype should be as similar as possible to the operational system.
- *The functionality of the system* Here the precise way the system should function internally is not known. For example, a computer model of some real-world phenomenon is being developed. The algorithms used might need to be repeatedly adjusted until they satisfactorily imitate real-world behaviour.

EXERCISE 4.4

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?

(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 taking phone calls from members of the public enquiring about motor insurance and giving 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.

Controlling changes during prototyping

A major problem with prototyping is controlling changes to the prototype following suggestions by the user. One approach has been to categorize changes as belonging to one of three types:

- *Cosmetic* (often about 35% of changes)
  These are simple changes to the layout of the screens or reports. They are:
  (a) implemented;
  (b) recorded.
- *Local* (often about 60% of changes)
  These change the way that a screen or report is processed but do not affect other parts of the system. They are:
  (a) implemented;
  (b) recorded;
  (c) backed-up so that they can be removed at a later stage if necessary;
  (d) inspected retrospectively.
- *Global* (about 5% of changes)
  These are changes that affect more than one part of the processing. All changes here have to be the subject of a design review before they can be implemented.

Inspections are discussed in Chapter 13.

4.11 Incremental Delivery

This approach breaks the application down into small components which are then implemented and delivered in sequence. Each component delivered must give some benefit to the user. Figure 4.4 gives a general view of the approach.

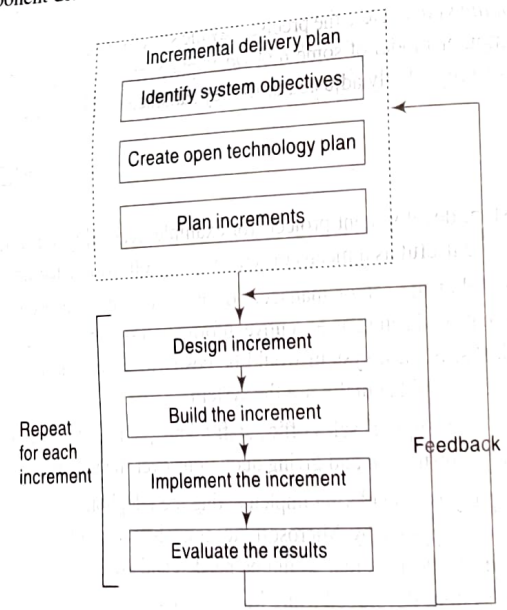


FIGURE 4.4 Intentional incremental delivery

Time-boxing is often associated with an incremental approach. Here the scope of deliverables for an increment is rigidly constrained by an agreed deadline. This deadline has to be met, even at the expense of dropping some of the planned functionality. Omitted features can be transferred to later increments.

Advantages of this approach

These are some of the justifications given for the approach.

- The feedback from early increments improves the later stages.
- The possibility of changes in requirements is reduced because of the shorter time span 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*, that is, the requesting of features that are unnecessary and not in fact used, is less as users know that 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 emerges.
- Job satisfaction is increased for developers who see their labours bearing fruit at regular, short, intervals.

Tom Gilb, whose *Principles of Software Engineering Management* was published by Addison-Wesley in 1988, is a prominent advocate of this approach.

Disadvantages

On the other hand, these disadvantages have been put forward.

- Later increments might require modifications to earlier increments. This is known as *software breakage*.
- Software developers may be more productive working on one large system than on a series of smaller ones.
- Grady Booch, an authority on OO, suggests that with what he calls 'requirements driven' projects (which equate to incremental delivery) *'Conceptual integrity sometimes suffers because there is little motivation to deal with scalability, extensibility, portability or reusability beyond what any vague requirements might imply.'* Booch also suggests there could be a tendency towards a large number of discrete functions with little common infrastructure.

This quotation is from Grady Booch (1996) *Object Solutions: Managing the Object Oriented Project*, Addison-Wesley.

The incremental delivery plan

The nature and order of each increment to be delivered to the users have to be planned at the outset.

This process is similar to strategic planning but at a more detailed level. 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*.

The process of planning the increments of a project as described by Gilb has similarities with strategic planning described in Chapter 2.

System objectives

Recall that earlier we suggested that project planners ideally want well-defined objectives but as much freedom as possible about how these are to be met. These overall objectives can 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. If this is done properly these overarching quality requirements can go some way to meeting the concerns, expressed by Grady Booch, that things might get lost with the concentration on the requirements at increment level. It also 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 changing environment of an application individual requirements could change over the course of the project, but the objectives should not.

Chapter 13 discusses software quality characteristics.

Open technology plan

If the system is to be able to cope with new components being continually added then it needs to be 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 the names of an organization and its departments, charge rates, and so on, 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 software development environment.

Although Gilb does not suggest this, following Booch's hints it would be desirable to draw up an initial logical data model or object model for the whole system. It is difficult to see how the next stage of planning the scope and order of each increment could be done without this foundation.

Incremental plan

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

- Steps typically should consist of 1–5% of the total project.
- Non-computer steps should be included.
- An increment should, ideally, not exceed one month and should not, at worst, take more than three months.

- Each increment should deliver some benefit to the user.
- Some increments will be physically dependent on others.
- In other cases value-to-cost ratios may be used to decide priorities (see below).

A non-computer step could be something like a streamlined clerical procedure.

A new system might be replacing an old computer system and the first increments could use parts of the old system. For example, the data for the database of the new system could initially be obtained from the old system's standing files.

Which steps should be first? Some steps will be prerequisites because of physical dependencies but others can be in any order. Value-to-cost ratios (see Table 4.1) can be used to establish the order in which increments are to be developed. The customer is asked to rate the value of each increment with a score in the range 1–10. The developers also rate the cost of developing each of the increments with a score in the range 0–10. This might seem rather crude, but people are often unwilling to be more precise. Dividing the value rating by the cost rating generates a ratio which indicates the relative 'value for money' of each increment.

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)

A zero cost would mean that the change can be implemented without software development – some costs might be incurred by users in changing procedures.

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.

An incremental example

Tom Gilb describes a project where a software supplier 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 the real effort.

The project was replanned and 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 application had actually been delivered early.

4.12 Atern/Dynamic Systems Development Method

In the United Kingdom, SSADM (Structured Systems Analysis and Design Method) has until recently been a predominant methodology. In no small part, this has been because of sponsorship by the United Kingdom government. More recently, however, it has lost some favour, partly because it has been perceived as overly

bureaucratic and prescriptive. In contrast, there has been an increased interest in the iterative and incremental approaches we have outlined above. As a consequence, a consortium has developed guidelines for the use of such techniques and packaged the overall approach as the Dynamic Systems Development Method (DSDM). This has been re-badged as Atern. It is possible to attend courses on the method and to become an accredited Atern practitioner.

Eight core Atern principles have been enunciated.

1. *Focus on business need*. Every decision in the development process should be taken with a view to best satisfying business needs. Effectively this is emphasizing the need to avoid means–end inversion that we described in Section 4.1, that is, focusing on the detail of a procedure to the detriment of satisfactory project deliverables.
2. *Deliver on time*. Time-boxing is applied. Every deadline will see the delivery of valuable products, even if some less valuable ones are held over. This is better than delivery dates being pushed back until a delivery of all scheduled products can be made.
3. *Collaborate*. A one-team culture should be promoted, where user representatives are integrated into the delivery team.
4. *Never compromise quality*. Realistic quality targets are set early in the project. A process of continuously testing developing products starting as soon as possible is adopted.
5. *Develop iteratively*. The prototyping approach described in Section 4.8 would be an example of how this might be done.
6. *Build incrementally from firm foundations*. The incremental delivery approach as described in Section 4.11 is embraced.
7. *Communicate continuously*. In the case of users this could, for example, be done via workshops and the demonstration of prototypes.
8. *Demonstrate control*. Atern methodology has a range of plans and reports that can be used to communicate project intentions and outcomes to project sponsors and other management stakeholders.

JAD, Joint Application Development, was discussed in Section 4.6.

Figure 4.5 outlines the general approach. The main life cycle phases are shown:

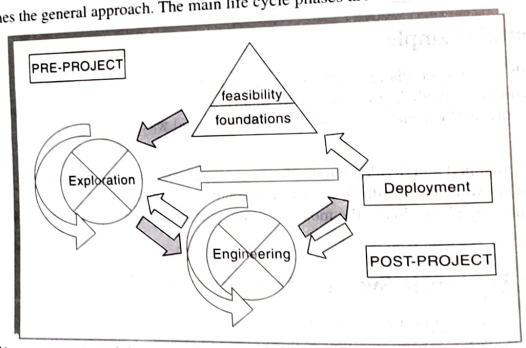


FIGURE 4.5 Atern process model

- *Feasibility/foundation*. Among the activities undertaken here is derivation of a business case of the sort discussed in Chapter 2 and general outlines of the proposed architecture of the system to be developed.
- *Exploration cycle*. This investigates the business requirements. These requirements are translated into a viable design for the application. This could be an iterative process that could involve the creation of exploratory prototypes. A large project could be decomposed into smaller increments to assist the design process.
- *Engineering cycle*. This takes the design generated in the exploration cycle and converts it into usable components of the final system that will be used operationally. Once again this could be done using incremental and evolutionary techniques.
- *Deployment*. This gets the application created in the engineering cycle into actual operational use.

Not only can there be iterations within the exploration and engineering cycles, but an increment could involve requirements investigation followed by the building of the functionality.

Atern encourages the use of time-boxes. It is suggested that these should typically be between two and six weeks in order to make participants focus on real needs. It will be recalled that in order to meet the deadline imposed by a time-box, the implementation of less important features may be held over to later increments (or even dropped altogether). The relative importance of requirements can be categorized using the 'MoSCoW' classification:

Time-boxes were discussed in Section 4.11 on incremental delivery.

- *Must have*: that is, essential features.
- *Should have*: these would probably be mandatory if you were using a conventional development approach – but the system can operate without them.
- *Could have*: these requirements can be delayed with some inconvenience.
- *Won't have*: these features are wanted, but their delay to a later increment is readily accepted.

The possibility of requirements being reallocated to different increments means that project plans will need to be constantly updated if the project is to be successfully controlled.

4.13 Rapid Application Development

Rapid Application Development (RAD) model is also sometimes referred to as the rapid prototyping model. This model has the features of both the prototyping and the incremental delivery models.

The major aims of the RAD model are as follows:

- to decrease the time taken and the cost incurred to develop software systems; and
- to limit the costs of accommodating change requests by incorporating them as early as possible before large investments have been made on development and testing.

One of the major problems that has been identified with the waterfall model is the following. Clients often do not know what they exactly want until they see a working system. However, they do not see the working system until the development is complete in all respects and delivered to them. As a result, the exact requirements are brought out only through the process of customers commenting on the installed application. The required changes are incorporated through subsequent maintenance efforts. This makes the cost of accommodating any change extremely high. As a result, it takes a long time and enormous cost to have a good solution

in place. Clearly, this model of developing software would displease even the best customer. The RAD model tries to overcome this problem by inviting and incorporating customer feedback on successively developed prototypes. In the RAD model, absence of long-term and detailed planning gives the flexibility to accommodate requirements change requests solicited from the customer during project execution.

In the RAD model, development takes place in a series of short cycles called iterations. Plans are made for one iteration at a time only. The time planned for each iteration is called a time box. Each iteration enhances the implemented functionality of the application a little. During each iteration, a quick and dirty prototype for some functionality is developed. The customer evaluates the prototype and gives feedback, based on which the prototype is refined. Thus, over successive iterations, the full set of functionalities of the software takes shape. However, it needs to be noted that in the RAD model, the prototype is used as an instrument for gathering customer feedback only and is not released to the customer for regular use.

The development team is also required to include a customer representative to clarify the requirements. Thus conscious attempts are made to bridge the communication gap between the customer and the development team and to tune the system to the exact customer requirements. But, how does RAD model lead to faster product development? RAD emphasizes code reuse as an important means to get the work done fast. In fact the RAD adopters were the earliest to embrace object-oriented languages and practices. RAD also advocates the use of specialized tools for faster creation of working prototypes.

4.14 Agile Methods

Agile methods are designed to overcome the disadvantages we have noted in our discussions on the heavyweight implementation methodologies. One of the main disadvantages of the traditional heavyweight methodologies is the difficulty of efficiently accommodating change requests from customers during execution of the project. Note that the agile model is an umbrella term that refers to a group of development processes, and not a single model of software development. There are various agile approaches such as the following:

- Crystal Technologies
- Atern (formerly DSDM)
- Feature-driven Development
- Scrum
- Extreme Programming (XP)

The Agile Manifesto is available at http://www.agilealliance.org

In the agile model, the feature requirements are decomposed into several small parts that can be incrementally developed. The agile model adopts an iterative approach. Each incremental part is developed over an iteration. Each iteration is intended to be small and easily manageable, and lasts for a couple of weeks only. At a time, only one increment is planned, developed, and then deployed at the customer's site. No long-term plans are made. The time taken to complete an iteration is called a time box. The implication of the term 'time box' is that the end date for an iteration does not change. The development team can, however, decide to reduce the delivered functionality during a time box, if necessary, but the delivery date is considered sacrosanct.

See S. Nerur, R. Mahapatra and G. Mangalara (2005) 'Challenges of migrating to agile methodologies' *Communications of the ACM* 48(5) 73–8.

Besides the delivery of increments after each time box, a few other principles discussed below are central to the agile model.

- Agile model emphasizes face-to-face communication over written documents. Team size is deliberately kept small (5–9 people) to help the team members effectively communicate with each other and collaborate. This makes the agile model well-suited to the development of small projects, though large projects can also be executed using the agile model. In a large project, it is likely that the collaborating teams might work at different locations. In this case, the different teams maintain daily contact through video conferencing, telephone, e-mail, etc.

See H. Merisalo-Rantanen, T. Tuure and M. Rossi (2005) 'Is extreme programming just old wine in new bottles?' *Journal of Database Management* 16(4) 41–61.

- An agile project usually includes a customer representative in the team. At the end of each iteration, the customer representative along with the stakeholders review the progress made, re-evaluate the requirements, and give suitable feedback to the development team.
- Agile development projects usually deploy pair programming. In this approach, two programmers work together at one work station. One types the code while the other reviews the code as it is typed. The two programmers switch their roles every hour or so. Several studies indicate that programmers working in pairs produce compact well-written programs and commit fewer errors as compared to programmers working alone.

EXERCISE 4.5

As can be seen, there is much in common between the RAD model and the agile model. Identify the important differences between the agile model and the RAD model.

4.15 Extreme Programming (XP)

See Kent Beck (with Cynthia Andreas), *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1st edition 1999, 2nd edition 2004. 'Extreme programming' is sometimes shown with a capital 'X' i.e. 'eXtreme Programming'.

The primary source of information on XP is Kent Beck's *Extreme programming explained: embrace change*, first published in 1999 and updated in 2004. The description here is based on the first edition, with some comments where the ideas have been developed further in the second.

The ideas were largely developed on the C3 payroll development project at Chrysler. The approach is called 'extreme programming' because, according to Beck, '*XP takes commonsense principles to extreme levels*'. Four core values are presented as the foundations of XP.

1. **Communication and feedback.** It is argued that the best method of communication is face-to-face communication. Also, the best way of communicating to the users the nature of the software under production is to provide them with frequent working increments. Formal documentation is avoided.
2. **Simplicity.** The simplest design that implements the users' requirements should always be adopted. Effort should not be spent trying to cater for future possible needs – which in any case might never actually materialize.
3. **Responsibility.** The developers are the ones who are ultimately responsible for the quality of the software – not, for example, some system testing or quality control group.
4. **Courage.** This is the courage to throw away work in which you have already invested a lot of effort, and to start with a fresh design if that is what is called for. It is also the courage to try out new ideas – after

all, if they do not work out, they can always be scrapped. Beck argues that this attitude is more likely to lead to better solutions.

Among the *core practices* of XP are the following.

The planning exercise

In the second edition of his book, Beck favours iterations of one week on the grounds that people tend to work naturally in weekly cycles.

Previously, when we talked about 'increments' we meant components of the system that users could actually use. XP refers to these as *releases*. Within these releases code is developed in *iterations*, periods of one to four weeks' duration during which specific features of the software are created. Note that these are not usually 'iterations' in the sense that they are new, improved, versions of the same feature – although this is a possibility. The planning game is the process whereby the features to be incorporated in the next release are negotiated. Each of the features is documented in a short textual description called a *story* that is written on a card. A process similar to value-to-cost ratio analysis discussed earlier in Section 4.11 or Atern's MoSCoW rating is carried out in order to give priorities to the features. At the time of the next code release, any features that have not been completed will be held over – that is, time-boxing is employed.

Small releases

The time between releases of functionality to users should be as short as possible. Beck suggests that releases should ideally take a month or two. This is compatible with Tom Gilb's recommendation of a month as the ideal time for an increment, with a maximum of three months.

Metaphor

The system to be built will be software code that reflects things that exist and happen in the real world. A payroll application will calculate and record payments to employees. The terms used to describe the corresponding software elements should, as far as possible, reflect real-world terminology – at a very basic level this would mean using meaningful names for variables and procedures such as 'hourly_rate' and 'calculate_gross_pay'. Beck suggests that what he calls the use of metaphor can do the job that 'system architecture' does on conventional projects. In this context 'architecture' refers to the use of system models such as class and collaboration diagrams to describe the system. The astute reader might point out that the use of the term 'architecture' is itself a metaphor.

Simple design

This is the practical implementation of the value of simplicity that was described above.

Testing

Testing is done at the same time as coding. The test inputs and expected results should be scripted so that the testing can be done using automated testing tools. These test cases can then be accumulated so that they can be used for regression testing to ensure that later developments do not insert errors into existing working code. This idea can be extended so that the tests and expected results are actually created before the code is created. Working out what tests are needed to check that a function is correct can itself help to clarify requirements. Two types of testing are needed: unit testing which focuses on the code a developer has just written, and function testing which is user-oriented and checks the correctness of a particular feature and which may involve several code units.

Refactoring

A threat to the target of striving to have always the simplest design is that over time, as modifications are made to code, the structure tends to become more spaghetti-like. The answer to this is to have the courage to resist the temptation to make changes that affect as little of the code as possible and be prepared to rewrite whole sections of code if this will keep the code structured. The repository of past test cases – see the section immediately above – can be executed to ensure that the refactoring has not introduced bugs into the application.

Pair programming

All software code is written by pairs of developers, one actually doing the typing and the other observing, discussing and making comments and suggestions about what the other is doing. At intervals, the developers can swap roles. The ideal is that you are constantly changing partners so that you get to know about a wide range of features that are under development. It follows from this that office environments need to be designed carefully to allow this type of working, and that developers will generally need to keep the same office hours.

Helen Sharp of the Open University has studied XP in practice. One of her observations is that the social nature of the development process encourages a rhythm of group meetings, pair working and daily 'builds' when new code is integrated that helps to give the project momentum. Interestingly, this rhythm of activity and review acting as a heart-beat of the project has also been noted in a successful dispersed project.

Collective ownership

This is really the corollary of pair programming. The team as a whole takes collective responsibility for the code in the system. A unit of code does not 'belong' to just one programmer who is the only one who can modify it.

Continuous integration

This is another aspect of testing practice. As changes are made to software units, integrated tests are run regularly – at least once a day – to ensure that all the components work together correctly.

Forty-hour weeks

Chapter 11 discusses, among other issues, the question of stress. It points out that working excessive hours (in some cases 60 hours or more a week) can lead to ill-health and be generally counterproductive. The principle is that normally developers should not work more than 40 hours a week. It is realistic to accept that sometimes there is a need for overtime work to deal with a particular problem – but in this case overtime should not be worked for two weeks in a row. Interestingly, in some case studies of the application of XP, the 40-hour rule was the only one not adhered to.

On-site customers

Fast and effective communication with the users is achieved by having a user domain expert on-site with the developers.

Coding standards

If code is genuinely to be shared, then there must be common, accepted, coding standards to support the understanding and ease of modification of the code.

Limitations of XP

The successful use of XP is based on certain conditions. If these do not exist, then its practice could be difficult. These conditions include the following.

- There must be easy access to users, or at least a customer representative who is a domain expert. This may be difficult where developers and users belong to different organizations.
- Development staff need to be physically located in the same office.
- As users find out about how the system will work only by being presented with working versions of the code, there may be communication problems if the application does not have a visual interface.
- For work to be sequenced into small iterations of work, it must be possible to break the system functionality into relatively small and self-contained components.
- Large, complex systems may initially need significant architectural effort. This might preclude the use of XP.

XP does also have some intrinsic potential problems – particularly with regard to its reliance on tacit experience and knowledge as opposed to externalized knowledge in the shape of documentation.

- There is a reliance on high-quality developers which makes software development vulnerable if staff turnover is significant.
- Even where staff retention is good, once an application has been developed and implemented, the tacit, personal, knowledge of the system may decay. This might make it difficult, for example, for maintenance staff without documentation to identify which bits of the code to modify to implement a change in requirements.
- Having a repository of comprehensive and accurate test data and expected results may not be as helpful as might be expected if the rationale for particular test cases is not documented. For example, when a change is made to the code, how do you know which test cases need to be changed?
- Some software development environments have focused on encouraging code reuse as a means of improving software development productivity. Such a policy would seem to be incompatible with XP.

4.16 Scrum

In the Scrum model, projects are divided into small parts of work that can be incrementally developed and delivered over time boxes that are called sprints. The product therefore gets developed over a series of manageable chunks. Each sprint typically takes only a couple of weeks. At the end of each sprint, stakeholders and team members meet to assess the progress and the stakeholders suggest to the development any changes and improvements they feel necessary.

In the scrum model, the team members assume three fundamental roles, viz., product owner, scrum master, and team member. The product owner is responsible for communicating the customer's vision of the product to the development team. The scrum master acts as a liaison between the product owner and the team, and facilitates the development work.

4.17 Managing Iterative Processes

This discussion of agile methods might be confusing as it seems to turn many of our previous planning concepts on their head.

Approaches like XP correctly emphasize the importance of communication and of removing artificial barriers to development productivity. XP to many might seem to be simply a 'licence to hack'. However, a more detailed examination of the techniques of XP shows that many (such as pair programming and installation standards) are conscious techniques to counter the excesses of hacking and to ensure that good maintainable code is written.

Booch suggests that there are two levels of development: the macro process and the micro process. The macro process is closely related to the waterfall process model. At this level, a range of activities carried out by a variety of specialist groups has to be coordinated. We need to have some dates when we know that major activities will be finished so that we know when we will need to bring in staff to work on subsequent activities. Within this macro process there will be micro process activities which might involve iterative working. Systems testing has always been one. Figure 4.6 illustrates how a sequential macro process can be imposed

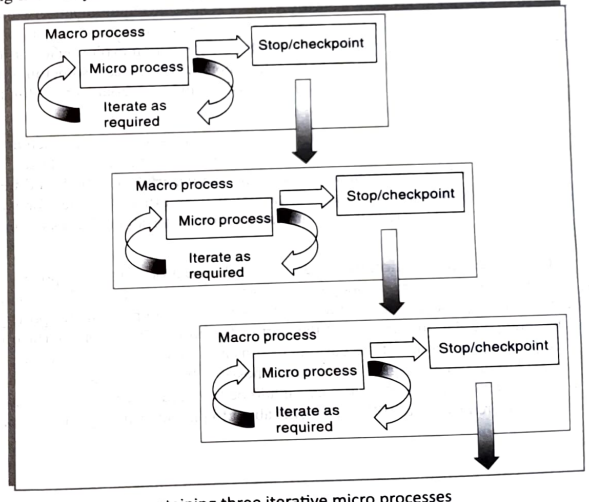


FIGURE 4.6 A macro process containing three iterative micro processes

on a number of iterative sub-processes. With iterative micro processes, the use of time-boxes is needed to control at the macro level.

There are cases where the macro process itself can be iterative. It might be that a prototype for a complex technical system is produced in two or three successive versions, each taking several months to create and evaluate. In these circumstances, each iteration should be treated as a project in its own right.

D. Karlström and P. Runeson (2005). 'Combining agile methods with stage-gate project management' *IEEE Software*, May/June.

The packaging of micro processes within larger macro processes means that it is possible for agile projects using XP practices to exist within a more traditional stage-gate project environment (see Section 4.7) which has formal milestones where the business case for the project is reviewed. Agile projects might even contribute helpfully to this process as their progress is more visible.

4.18 Selecting the Most Appropriate Process Model

Construction of an application can be distinguished from its *installation*. It is possible to use different approaches for these two stages. For example, an application could be constructed using a waterfall or 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 the evolutionary installation with any construction approach other than evolutionary.

Whenever uncertainty is high, an evolutionary approach needs to be favoured. An example of this situation would be where the users' requirements are not clearly defined. Where the requirements are relatively certain but there are many complexities, as with a large embedded system needing a large amount of code, then an incremental approach is 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.

Selection of an appropriate process model for a project can depend on several issues such as the characteristics of the software to be developed, the characteristics of the development team, and those of the customer. For development of simple and well-understood applications, the waterfall model should be sufficient. In fact, the waterfall model should be preferred in this case as it is an efficient model for well-understood projects undertaken by experienced programmers. Furthermore, the waterfall model results in production of good documents.

If the development team is entirely novice, then even the development of a simple application may require an incremental or prototyping model to be adopted. An incremental delivery model is usually suitable for object-oriented development projects. The spiral model would be appropriate, if the project is large and it is not possible to anticipate the project risks at the start of the project. The user interface part is usually developed using the prototyping model. If the customer is unsure about some features of the software to be developed, then an evolutionary or an agile model would possibly be the best choice. Where deadlines are tight, an evolutionary, incremental, or agile approach should be favoured over a one-shot strategy, as at least something would get delivered by the deadline. Students about to plan their final-year projects should make a note of this.

EXERCISE 4.6

A travel agency needs software for automating its book-keeping activities. The set of activities to be automated are rather simple and are at present being carried out manually. The travel agency has indicated that it is unsure about the type of user interface which would be suitable for its employees and its customers. Would it be proper for a development team to use the spiral model for developing this software?

CONCLUSION

This chapter has stressed the need to examine each project carefully to see if it has characteristics which suggest a particular approach or process model. These characteristics might 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 may be able to reduce 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.

FURTHER EXERCISES

1. A building society 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 alternative approaches should be considered?
   (b) Outline a plan for the development of the system which illustrates the application of your preferred methodology for this project.
2. A software package is to be designed and built to assist in software cost estimation. It will input certain parameters and produce 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 transactions: amend invoice, produce invoice, produce monthly statements, record cash payment, clear paid invoices from database, create customer record, 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 users (**Hint** – think about the problems of taking existing details onto a database when a system is first implemented)?

4. In Section 4.10 the need was stressed of defining what is to be learnt from a prototype and the way that it will be evaluated to obtain the new knowledge. Outline the learning outcomes and evaluation for the following.
   (a) A final-year degree student is to build an application that will act as a 'suggestions box' in a factory. The application will allow employees to make suggestions about process improvements, and will track the subsequent progress of the suggestion as it is evaluated. The student wants to use a web-based front-end with a conventional database. The student has not previously developed any applications using this mix of technologies.
   (b) An engineering company has to maintain a large number of different types of document relating to current and previous projects. It has decided to evaluate the use of a computer-based document retrieval system and wishes to try it out on a trial basis.
   (c) A business which specializes in 'e-solutions', that is, the development of business applications that exploit the World Wide Web, has been approached by the computing school of a local university. The school is investigating setting up a special website for its former students. The website's core will be information about job and training opportunities and it is hoped that this will generate income through advertising. It is agreed that some kind of pilot to evaluate the scheme is needed.
5. In a college environment, an intranet for students that holds information about courses, such as lecture programmes, reading lists and assignment briefs, is often set up. As a 'real' exercise, plan, organize and carry out a JAD session to design (or improve the design of) an intranet facility.

   This will require:
   - preliminary interviews with representative key stakeholders (for example, staff who might be supplying information for the intranet);
   - creation of documents for use in the JAD proceedings;
   - recording of the JAD proceedings;
   - creating a report which will present the findings of the JAD session.
6. What are the major shortcomings of the waterfall model? How have those shortcomings been overcome by the agile model?
7. Identify the pros and cons of using pair programming over programmers working alone. Based on your analysis, point out if there are any situations where the pair programming technique may not be suitable.

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;
- estimate the effort needed to implement software using a procedural programming language;
- count the function points for a system;
- understand the COCOMO II approach to developing effort models.

5.1 Introduction

A successful project is one delivered '*on time, within budget and with the required quality*'. This implies that targets are set which the project manager then tries to meet. This assumes that the targets are reasonable – no account is taken of the possibility of project managers achieving record levels of productivity from their teams, but still not meeting a deadline because of incorrect initial estimates. Realistic estimates are therefore crucial.

A project manager like Amanda has to produce estimates of *effort*, which affect costs, and of activity *duration* which affect the delivery time. These could be different, as in the case where two testers work on the same task for the same five days.

Some of the difficulties of estimating arise from the complexity and invisibility of software. Also, the intensely human activities which 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.

- *Subjective nature of estimating* For example, some research shows that people tend to underestimate the difficulty of small tasks and over-estimate that of large ones.
- *Political implications* Different groups within an organization have different objectives. The IOE information systems development managers may, for example, want to generate work and will pressurize estimators to reduce cost estimates to encourage higher management to approve projects. As Amanda, responsible for the development of the annual maintenance contracts subsystem, she will want to ensure that the project is within budget and timescale, otherwise this will reflect badly on herself. She might therefore try to increase the estimates to create 'comfort zone'. To avoid these 'political' influences, one suggestion is that estimates be produced by a specialist estimating group, independent of the users and the project team. Not all agree with this, as developers will be more committed to targets they themselves have set.

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

- *Changing technology* Where technologies change rapidly, it is difficult to use the experience of previous projects on new ones.
- *Lack of homogeneity of project experience* Even where technologies have not changed, knowledge about typical task durations may not be easily transferred from one project to another because of other differences between projects.

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 between the phases of design, coding and testing that could be expected from a new project.

The ISO 12207 standard, touched upon in Chapter 1, is an attempt to address this problem by standardizing on some of the terms used.

Using existing project data for estimating is also difficult because of uncertainties in the way that various terms can be interpreted. For example, what exactly is meant by the term 'testing'? Does it cover the activities of the software developer when debugging code?

EXERCISE 5.1

Calculate the productivity (i.e. 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?

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

Project	Design		Coding		Testing		Total	
	wm	(%)	wm	(%)	wm	(%)	wm	SLOC
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

(Contd)

(Contd)

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 (1985) 'Software project development cost estimation' *Journal of Systems and Software (5)*. The abbreviation SLOC stands for 'source lines of code'. SLOC is one way of indicating the size of a system.

EXERCISE 5.2

In the data presented in Table 5.1, observe that programmer productivity varies from 7 SLOC/day to 150 SLOC/day. In fact, in the industry the average productivity figure for programmers is only about 10 SLOC/day. Would you consider programmer productivity of 10 SLOC/day to be too low?

5.2 Where are Estimates Done?

Estimates are carried out at various stages of a software project for a variety of reasons.

- *Strategic planning* Project portfolio management involves estimating the costs and benefits of new applications in order to allocate priorities. Such estimates may also influence the scale of development staff recruitment.

Chapter 2 discussed project portfolio management in some detail.

- *Feasibility study* This confirms 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 which shows 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.
- *Evaluation of suppliers' proposals* In the case of the IOE annual maintenance contracts subsystem, for example, IOE might consider putting development out to tender. Potential contractors would scrutinize the system specification and produce estimates as the basis of their bids. Amanda might still produce her own estimates so that IOE could question a proposal which seems too low in order to ensure that the proposer has properly understood the requirements. The cost of bids could also be compared to in-house development.

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

- *Project planning* As the planning and implementation of the project becomes more detailed, more estimates of smaller work components will be made. These will confirm earlier broad-brush estimates, and will support more detailed planning, especially staff allocations.

As the project proceeds, so the accuracy of the estimates should improve as knowledge about the project increases. At the beginning of the project the user requirement is of paramount importance and premature consideration of the possible physical implementation is discouraged. However, in order to produce an estimate, there will need to be speculation about the eventual shape of the application.

To set estimating into the context of the Step Wise framework (Figure 5.1) presented in Chapter 3, re-estimating could take place at almost any step, but specific provision is made for the production of a relatively high-level estimate at Step 3, 'Analyse project characteristics', and for each individual activity in Step 5. 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.

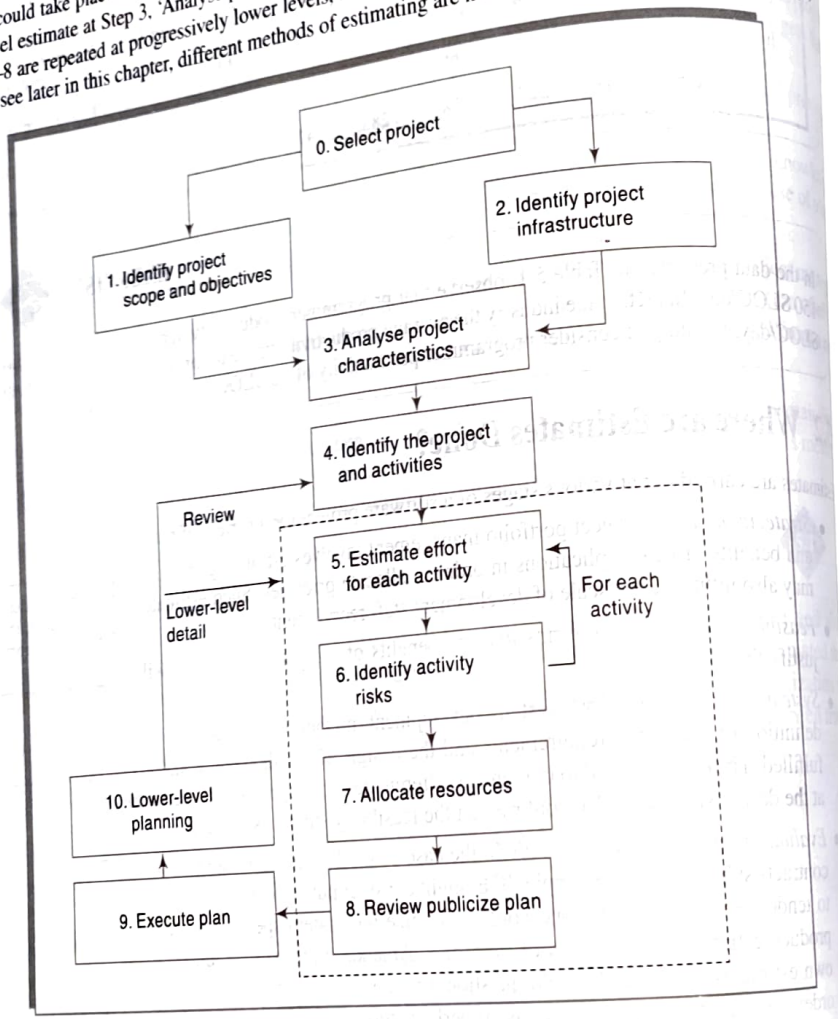


FIGURE 5.1 Software estimation takes place in Steps 3 and 5 in particular

5.3 Problems with Over- and Under-Estimates

A project leader such as Amanda will need to be aware that an over-estimate may 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*') that is, given an easy target staff will work less hard.
- *Brooks' Law* The effort of implementing 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, coordination and communication. This has given rise, in extreme cases, to the notion of Brooks' Law: '*putting more people on a late job makes it later*'. If there is an over-estimate of the effort required, this could lead to more staff being allocated than needed and managerial overheads being increased.

Parkinson's law was originally expounded in C. Northcote Parkinson's tongue-in-cheek book *Parkinson's Law*, John Murray, 1957. Brooks' law comes from *The Mythical Man-month* which has been referred to already.

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.

The danger with the under-estimate is the effect on quality. Staff, particularly those with less experience, could respond to pressing deadlines by producing work that is substandard. This may be seen as a manifestation of Weinberg's zeroth law of reliability: '*if a system does not have to be reliable, it can meet any other objective*'. Substandard work might only become visible at the later, testing, phases of a project which are particularly difficult to control and where extensive rework can easily delay project completion.

See, for example, T. K. Hamid and S. E. Madnick (1986) 'Impact of schedule estimation on software project behaviour' *IEEE Software* July 3(4) 70–5.

EXERCISE 5.3

How do agile methods such as XP – see Chapter 4 – attempt to address the problems with estimates described above?

Research has found that motivation and morale are enhanced where targets are achievable. If, over time, staff become aware that the targets set are unattainable and that projects routinely miss targets, motivation is reduced. People like to think of themselves as winners and there is a general tendency to put success down to our own efforts and blame failure on the organization.

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 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

Most estimating methods need information about past projects. However, care is needed when applying past performance to new projects because of possible differences in factors such as programming languages and the experience of staff. If past project data is lacking, externally maintained datasets of project performance data can be accessed. One well-known international database is that maintained by the International Software Benchmarking Standards Group (ISBSG), which currently contains data from 4800 projects.

Details of the work of the International Software Benchmarking Standards Group can be found at http://isbsg.org

Parameters to be estimated

The project manager needs to estimate two project parameters for carrying out project planning. These two parameters are effort and duration. Duration is usually measured in months. Work-month (wm) is a popular unit for effort measurement. We have already used this unit of effort measurement in Table 5.1. The term person-month (pm) is also frequently used to mean the same as work-month. One person-month is the effort an individual can typically put in a month. The person-month estimate implicitly takes into account the productivity losses that normally occur due to time lost in holidays, weekly offs, coffee breaks, etc. Person-month (pm) is considered to be an appropriate unit for measuring effort compared to person-days or person-years because developers are typically assigned to a project for a certain number of months.

Measure of work

Measure of work involved in completing a project is also called the size of the project. Work itself can be characterized by cost in accomplishing the project and the time over which it is to be completed. Direct calculation of cost or time is difficult at the early stages of planning. The time taken to write the software may vary according to the competence or experience of the software developers might not even have been identified. Implementation time may also vary depending on the extent to which CASE (Computer Aided Software Engineering) tools are used during development. It is therefore a standard practice to first estimate the project size; and by using it, the effort and time taken to develop the software can be computed. Thus, we can consider project size as an independent variable and the effort or time required to develop the software as dependent variables.

Let us examine the meaning of the term 'project size'. The size of a project is obviously not the number of bytes that the source code occupies, neither is it the size of the executable code. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product. Two metrics are at present popularly being used to measure size. These are Source Lines of Code (SLOC) and Function Point (FP). The SLOC measure suffers from various types of disadvantages, which are to a great extent corrected in the FP measure. However, the SLOC measure is intuitively simpler, so it is still being widely used. It is important, however, to be aware of the major shortcomings of the SLOC measure.

- *No precise definition.* SLOC is a very imprecise measure. Unfortunately, researchers have not been consistent on points like does it include comment lines or are data declarations to be included? The writers' view is that comment lines are excluded in determining the SLOC measure. This can be debated but the main point is that consistency is essential.

- *Difficult to estimate at start of a project.* From the project manager's perspective, the biggest shortcoming of the SLOC metric is that it is very difficult to estimate it during project planning stage, and can be accurately computed only after the development of the software is complete. The SLOC count can only be guessed at the beginning of a project, often leading to grossly inaccurate estimations.
- *Only a code measure.* SLOC is a measure of coding activity alone. This point has been illustrated in Exercise 5.2. A good problem size measure should consider the effort required for carrying out all the life cycle activities and not just coding.
- *Programmer-dependent.* SLOC gives a numerical value to the problem size that can vary widely with the coding style of individual programmers. This aspect alone renders any LOC-based size and effort estimations inaccurate.
- *Does not consider code complexity.* Two software components with the same KLOC will not necessarily take the same time to write, even if done by the same programmer in the same environment. One component might be more complex. Because of this, the effort estimate based on SLOC might have to be modified to take its complexity into account. Attempts have been made to find objective measures of complexity, but it depends to a large extent on the subjective judgment of the estimator.

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.

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*, based on the advice of knowledgeable staff;
- *Analogy*, where a similar, completed, project is identified and its actual effort is used as the basis of the estimate;
- *Parkinson*, where the staff effort available to do a project becomes the 'estimate';
- *price to win*, where the 'estimate' is a figure that seems sufficiently low to win a contract;
- *top-down*, where an overall estimate for the whole project is 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 (1981) *Software Engineering Economics*, Prentice-Hall.

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 identifying a price and not a prediction. Although Boehm rejects them as prediction techniques, they have value as management techniques. There is, for example, a perfectly acceptable engineering practice of '*design to cost*'.

This is also the principle behind the concept of time-boxing discussed in Chapter 4 in the context of incremental delivery.

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

5.6 Bottom-up Estimating

With the bottom-up approach the estimator breaks the project into its component tasks. With a large project the process of breaking it down into tasks is iterative: each task is decomposed into its component subtasks and these in turn could be further analysed. It is suggested that this is repeated until you get tasks an individual could do in a week or two. Why is this not a 'top-down approach'? After all, you start from the top and work down. Although this top-down analysis is an essential precursor to bottom-up estimating, it is really a separate process – that of producing a work breakdown schedule (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 best at the later, more detailed, stages of project planning. If this method is used earlier, assumptions about the characteristics of the final system and project work methods will have to be made.

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

EXERCISE 5.4

Brigette at Brightmouth College has been told that there is a requirement, once 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 may be obtained from the payroll standing data. The number of hours that each member of staff spends teaching on each course may be obtained from standing files in a computer-based timetabling 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?

A procedural code-oriented approach

The bottom-up approach described above works at the level of activities. In software development a major activity is writing code. Here we describe how a bottom-up approach can be used at the level of software components.

(a) **Envisage the number and type of software modules in the final system**
Most information systems, for example, are built from a small set of system operations, e.g. Add, Amend, Update, Display, Delete, Print. The same principle should equally apply to embedded systems, albeit with a different set of primitive functions.

'Software module' here implies a component that can be separately compiled and executed.

(b) **Estimate the SLOC of each identified module**
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 will be needed to implement each identified procedure. The estimator may look at existing programs which have a similar functional description to assist in the process.

(c) **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.

(d) **Calculate the work-days effort**
Historical data can be used to provide ratios to convert weighted SLOC to effort.

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 described later.

EXERCISE 5.5

The IOE annual maintenance contracts subsystem for which Amanda is responsible will have a transaction which sets up details of new annual maintenance contract customers.

The operator will input:

Customer account number
Customer name
Address
Postcode
Customer type
Renewal date

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.

Draw up an outline program structure diagram for a program to do the processing described above. For each box on your diagram, estimate the number of lines of code needed to implement the routine in a programming language that you are familiar with, such as Java.

5.7 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 is of practical concern to house-owners who need insurance cover to rebuild their property if destroyed. Unless the house-owner is in the building trade he or she is unlikely to be able to calculate the numbers of bricklayer-hours, carpenter-hours, electrician-hours, and so on, required. Insurance companies, however, produce convenient tables where the house-owner can find estimates of rebuilding costs based on such *parameters* as the number of storeys and the floor space of a house. This is a simple parametric model.

Project effort relates mainly to variables associated with characteristics of the final system. A parametric model will normally have one or more formulae in the form:

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

For example, system size might be in the form 'thousands of lines of code' (KLOC) and have the specific value of 3 KLOC while the productivity rate was 40 days per KLOC. These values will often be matters of judgement.

A model to forecast software development effort therefore has two key components. The first is a method of assessing the amount of the work needed. The second assesses the rate of work at which the task can be done. For example, Amanda at IOE may estimate that the first software module to be constructed is 2 KLOC. She may 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 per day and complete the work in 2 × 40 days, i.e. 80 days, while Ken, who is less experienced, would need 55 days per KLOC and take 2 × 55, i.e. 110 days to complete the task. In this case KLOC is a size driver, indicating the amount of work to be done, while developer experience is a productivity driver influencing the productivity or work rate.

If you have figures for the effort expended on past projects (in work-days for instance) and also the system sizes in KLOC, you should be able to work out a productivity rate as

$$productivity = effort/size$$

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

$$effort = constant_1 + (size \times constant_2)$$

Some parametric models, such as that implied by function points, are focused on system or task size, while others, such are COCOMO, are more concerned with productivity factors. Those particular models are described in more detail later in this chapter.

EXERCISE 5.6

Students on a course are required to produce a written report on an ICT-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?

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.

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.

5.8 Expert Judgement

This is asking for an estimate of task effort from someone who is knowledgeable about either the application or the development environment. This method is often used when estimating the effort needed to change an existing piece of software. The estimator would have to examine the existing code in order to judge the proportion of code 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), supplemented by bottom-up estimating.

See R. T. Hughes (1996) 'Expert judgement as an estimating method' Information and Software Technology 38(3) 67–75.

There may be cases where the opinions of more than one expert may need to be combined. The Delphi technique described in Section 12.3 tackles group decision-making.

5.9 Estimating by Analogy

This is also called *case-based reasoning*. The estimator identifies completed projects (*source cases*) with similar characteristics to the new project (the *target case*). The effort recorded for the matching source case is then used as a base estimate for the target. The estimator then identifies differences between the target and the source and adjusts the base estimate to produce an estimate for the new project.

See M. Shepperd and C. Schofield (1997) 'Estimating software project effort using analogies' *IEEE Transactions in Software Engineering* SE-23(11) 736–43.

This can be a good approach where you have information about some previous projects but not enough to draw generalized conclusions about what might be useful drivers or typical productivity rates.

A problem is identifying the similarities and differences between applications where you have a large number of past projects to analyse. One attempt to automate this selection process is the ANGEL software tool. This identifies the source case that is nearest the target by measuring the Euclidean distance between cases. The Euclidean distance is calculated as:

$$distance = square\text{-}root\ of\ ((target\_parameter_1 - source\_parameter_1)^2 + \ldots (target\_parameter_n - source\_parameter_n)^2)$$

5.1 EXAMPLE

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 application 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 5.7

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 may be calculated. The ANGEL package uses rather more sophisticated algorithms based on this principle.

5.10 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 to quantify the functional size of programs independently of their programming languages. He developed the idea of function points (FPs).

The basis of function point analysis is that 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 which update internal computer files.
- *External output types* are transactions where data is output to the user. Typically these would be printed reports, as screen displays would tend to come under external inquiry types (see below).
- *External inquiry types* – note the US spelling of inquiry – are transactions initiated by the user which provide information but do not update the internal files. The user inputs some information that directs the system to the details required.
- *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 items that is usually accessed together. It may be made up of one or more *record types*. For example, a purchase order file may be made up of a record type PURCHASE-ORDER plus a second which is repeated for each item ordered on the purchase order – PURCHASE-ORDER-ITEM. In structured systems analysis, a logical internal file would equate to a datastore, while record types would equate to relational tables or entity types.
- *External interface file types* allow for output and input that may 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 between applications would also be counted here.

The analyst identifies each instance of each external user type in the application. 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.) (1993) *Software Engineering Metrics* (Vol. 1), McGraw-Hill.

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

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
External inquiry type	3	4	6
Logical internal file type	7	10	15
External interface file type	5	7	10

EXERCISE 5.8

The task for which Brigette has been made responsible in Exercise 5.4 needs a program which will extract yearly salaries from the payroll file, and hours taught on each course by each member of staff and the details of courses from two files maintained by the timetabling system. The program will 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.

With FPs as originally defined by Albrecht, the question of whether the external user type was of high, low or average complexity was intuitive. The International FP User Group (IFPUG) has now promulgated rules on how this is assessed. 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. Similar tables exist for external inputs and outputs.

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 5.3 IFPUG file type complexity

Number of record types	Number of data types		
	<20	20–50	>50
1	Low	Low	Average
2 to 5	Low	Average	High
>5	Average	High	High

5.2 EXAMPLE

A logical internal file might contain data about purchase orders. These purchase orders might be organized into two separate record types: the main PURCHASE-ORDER details, namely purchase order number, supplier reference and purchase order date, and then details for each PURCHASE-ORDER-ITEM 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 7 for this file.

Function point analysis recognizes that the effort required to implement a computer-based information system relates not just to the number and complexity of the features provided but also to the operational environment of the system.

Fourteen factors have been identified which can influence the degree of difficulty associated with implementing a system. The list that Albrecht produced related particularly to the concerns of information system

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

developers in the late 1970s and early 1980s. Some technology which was then new and relatively threatening is now well established.

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

The COCOMO II Model Definition Manual A contains a table of suggested conversion rates and can be downloaded from http://sunset.usc.edu/csse

Tables have been calculated to convert the FPs to lines of code for various languages. For example, it is suggested that 53 lines of Java are needed on average to implement an FP, while for Visual C++ the figure is 34. You can then use historical productivity data to convert the lines of code into an effort estimate, as previously described in Section 5.7.

EXERCISE 5.9

In the case of the subsystem described in Exercise 5.8 for which Brigette is responsible at Brightmouth HE College, how many lines of Java code should be needed to implement this subsystem, according to the standard conversion? Assuming a productivity rate of 50 lines of code a day, what would be the estimate of effort?

5.11 Function Points Mark II

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

The Mark II method was originally sponsored by what was then the CCTA (Central Computer and Telecommunications Agency, now the Office of Government Commerce or OGC), which lays down standards for UK government projects. 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 United Kingdom.

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 is that an information system comprises transactions which have the basic structure shown in Figure 5.2.

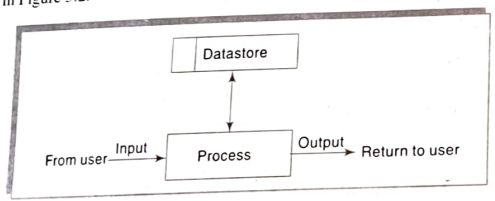


FIGURE 5.2 Model of a transaction

For each transaction the UFPs are calculated:

$W_i \times$ (*number of input data element types*) +

$W_e \times$ (*number of entity types referenced*) +

$W_o \times$ (*number of output data element types*)

W_i, W_e and W_o are weightings derived by asking developers the proportions of effort spent in previous projects developing the code dealing respectively with inputs, accessing and modifying stored data and processing outputs.

The proportions of effort are then normalized into ratios, or weightings, which add up to 2.5. This process for calculating weightings is time consuming and most FP counters use industry averages which are currently 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.

5.3 EXAMPLE

A cash receipt transaction in the IOE maintenance accounts subsystem accesses two entity types – INVOICE and CASH-RECEIPT.

The data inputs are:

Invoice number

Date received

Cash received

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 CASH-RECEIPT record is created. The error message is the only output of the transaction. The unadjusted function points, using the industry average weightings, for this transaction would therefore be:

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

EXERCISE 5.10

Calculate the number of unadjusted Mark II function points for the transaction described previously for Exercise 5.5, using the industry average weightings.

Mark II FPs follow the Albrecht method in recognizing that one system delivering the same functionality as another may 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 identification of further factors to suit local circumstances is encouraged.

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 by the size of the project. In general, larger projects, up to a certain point, are 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, have been found useful as a way of calculating the price for extensions to existing systems, as will be seen in Chapter 10 on managing contracts.

5.12 COSMIC Full Function Points

COSMIC-FFP stands for Common Software Measurement Consortium – Full Function Points.

While approaches like that of IFPUG are suitable for information systems, they are not helpful when it comes to sizing real-time or embedded applications. This has resulted in the development of another version of function points – the COSMIC FFP method.

The full function point (FFP) method has its origins in the work of two interlinked research groups in Québec, Canada. At the start, the developers were at pains to stress that this method should be seen as simply an extension to the IFPUG method for real-time systems. The original work of FFPs has been taken forward by the formation of the Common Software Measurement Consortium (COSMIC) which has involved not just the original developers in Canada, but others from many parts of the world, including Charles Symons, the originator of Mark II function points. Interestingly, there has been little participation by anyone from the United States.

The argument is that existing function point methods are effective in assessing the work content of information systems where the size of the internal procedures mirrors the number of external features. With a real-time, or embedded, system, its features will be hidden because the software's user will probably not be a human but a hardware device or another software component.

COSMIC deals with this by decomposing the system architecture into a hierarchy of software *layers*. The software component to be sized can receive requests for services from layers above and can request services from those below it. At the same time there could be separate software components at the same level that engage in *peer-to-peer communication*. This identifies the boundary of the software component to be assessed and thus the points at which it receives inputs and transmits outputs. Inputs and outputs are aggregated into *data groups*, where each group brings together data items that relate to the same object of interest.

Data groups can be moved about in four ways:

- *entries* (E), which are effected by subprocesses that move the data group into the software component in question from a 'user' outside its boundary – this could be from another layer or another separate software component in the same layer via peer-to-peer communication;
- *exits* (X), which are effected by subprocesses that move the data group from the software component to a 'user' outside its boundary;
- *reads* (R), which are data movements that move data groups from persistent storage (such as a database) into the software component;
- *writes* (W), which are data movements that transfer data groups from the software component into persistent storage.

EXERCISE 5.11

A small computer system controls the entry of vehicles to a car park. Each time a vehicle pulls up before an entry barrier, a sensor notifies the computer system of the vehicle's presence. The system examines a count that it maintains of the number of vehicles that are currently in the car park. This count is kept on backing storage so that it will still be available if the system is temporarily shut down, for example because of a power cut. If the count does not exceed the maximum allowed then the barrier is lifted and the count is incremented. When a vehicle leaves the car park, a sensor detects the exit and reduces the count of vehicles.

There is a system administration system that can set the maximum number of cars allowed, and which can be used to adjust or replace the count of cars when the system is restarted.

Identify the entries, exits, reads and writes in this application.

The overall FFP count is derived by simply adding up the counts for each of the four types of data movement. The resulting units are *Cfsu* (COSMIC functional size units). The method does not take account of any processing of the data groups once they have been moved into the software component. The framers of the method do not recommend its use for systems involving complex mathematical algorithms, for example, but there is provision for the definition of local versions for specialized environments which could incorporate counts of other software features.

COSMIC FFPs have been incorporated into an ISO standard – ISO/IEC 19761:2003. Prior to this there were attempts to produce a single ISO standard for 'functional size measurement' and there is an ISO document – ISO/IEC 14143–1:1998 – which lays down some general principles. ISO has decided, diplomatically, that it is unable to judge the relative merits of the four main methods in the field: IFPUG, Mark II, NESMA and COSMIC-FFP, and all four have been allowed to submit their methods as candidates to become ISO standards and then to '*let the market decide*'.

The NESMA FP method has been developed by the Netherlands Software Measurement Association.

5.13 COCOMO II: A Parametric Productivity Model

Because there is now a newer COCOMO II, the older version is now referred to as COCOMO81.

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 the models could be used with applications other than information systems. The basic model was built around the equation

TABLE 5.4 COCOMO81 constants

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

$$(effort) = c(size)^k$$

where *effort* was measured in *pm* or the number of 'person-months' consisting of units of 152 working hours, *size* was measured in *kdsi*, thousands of delivered source code instructions, and c and k were constants.

The first step was to derive an estimate of the system size in terms of *kdsi*. The constants, c and k (see Table 5.4), 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.

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

- *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 that 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.

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 coordination.

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

Over the years, Barry Boehm and his co-workers have refined a family of cost estimation models of which the key one is COCOMO II. 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 has been 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 than previously. 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* Here 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* Here 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* Here the software structures undergo final construction, modification and tuning to create a system that will perform as required.

See R. D. Banker, R. Kauffman and R. Kumar (1992) 'An empirical test of object-based output measurement metrics' *Journal of MIS*, 8(3).

To estimate the effort for *application composition*, the counting of *object points* is recommended by the developers of COCOMO II. This follows the function point approach of counting externally apparent features of the software. It differs by focusing on the physical features of the application, such as screens and reports, rather than 'logical' ones such as entity types. This is seen as being more useful when the requirements are being elicited via prototypes.

At the *early design* stage, FPs are recommended as the way of gauging a basic system size. An FP count may be converted to an LOC equivalent by multiplying the FPs by a factor for the programming language that is to be used – see Section 5.10.

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

$$pm = A(size)^{(sf)} \times (em_1) \times (em_2) \times \ldots \times (em_n)$$

where *pm* is the effort in 'person-months', A is a constant (which was set in 2000 at 2.94), *size* is measured in *kdsi* (which may have been derived from an FP count as explained above), and *sf* is exponent scale factor.

The scale factor is derived thus:

$$sf = B + 0.01 \times \Sigma\ (exponent\ driver\ ratings)$$

where B is a constant currently set at 0.91. The effect of the exponent ('. . . to the power of. . .') scale factor is to increase the effort predicted for larger projects, that is, to take account of diseconomies of scale which make larger projects less productive.

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 increases the effort required disproportionately more on larger projects.

- *Precedentedness (PREC)* This quality is the degree to which there are precedents or similar past cases for the current project. The greater the novelty of the new system, the more uncertainty there is and the higher the value given to the exponent driver.

TABLE 5.5 COCOMO II Scale factor values

Driver	Very low	Low	Nominal	High	Very high	Extra high
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

- *Development flexibility (FLEX)* This reflects the number of different ways there are of meeting the requirements. The less flexibility there is, the higher the value of the exponent driver.
- *Architecture/risk resolution (RESL)* This reflects the degree of uncertainty about the requirements. If they are liable to change then a high value would be given to this exponent driver.
- *Team cohesion (TEAM)* 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 (PMAT)* Chapter 13 on software quality explains the process maturity model. The more structured and organized the way the software is produced, the lower the uncertainty and the lower the rating will be for this exponent driver.

Each of the scale factors for a project is rated according to a range of judgements: very low, low, nominal, high, very high, extra high. There is a number related to each rating of the individual scale factors – see Table 5.5. These are summed, then multiplied by 0.01 and added to the constant 0.91 to get the overall exponent scale factor.

EXERCISE 5.12

A new project has 'average' novelty for the software supplier that is going to execute it and is thus given a nominal rating on this account for precedentedness. Development flexibility is high, but requirements may change radically and so the risk resolution exponent is rated very low. The development team are all located in the same office and this leads to team cohesion being rated as very high, 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 rating of 'low'.

(i) What would be the scale factor (*sf*) in this case?

(ii) What would the estimate of effort if the size of the application was estimated as in the region of 2000 lines of code?

TABLE 5.6 COCOMO II Early design effort multipliers

Code	Effort modifier	Extra low	Very low	Low	Nominal	High	Very high	Extra high
RCPX	Product reliability and complexity	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE	Required reusability			0.95	1.00	1.07	1.15	1.24
PDIF	Platform difficulty			0.87	1.00	1.29	1.81	2.61
PERS	Personnel capability	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	Personnel experience	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	Facilities available	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED	Schedule pressure		1.43	1.14	1.00	1.00	1.00	

In the COCOMO II model the *effort multipliers* (*em*) adjust the estimate to take account of productivity factors, but do not involve economies or diseconomies of scale. The multipliers relevant to early design are in Table 5.6 and those used at the post architecture stage in Table 5.7. Each of these multipliers may, for a particular application, be given a rating of very low, low, nominal, high or very high. Each rating for each effort multiplier has an associated value. A value greater than 1 increases development effort, while a value less than 1 decreases it. The nominal rating means that the multiplier has no effect. The intention is that the values that these and other ratings use in COCOMO II will be refined over time as actual project details are added to the database.

TABLE 5.7 COCOMO II Post architecture effort multipliers

Modifier type	Code	Effort modifier
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
	SCED	Schedule pressure

EXERCISE 5.13

A software supplier has to produce an application that controls a piece of equipment in a factory. A high degree of reliability is needed as a malfunction could injure the operators. The algorithms to control the equipment are also complex. The product reliability and complexity are therefore rated as very high. The company would like to take the opportunity to exploit fully the investment that they made in the project by reusing the control system, with suitable modifications, on future contracts. The reusability requirement is therefore rated as very high. Developers are familiar with the platform and the possibility of potential problems in that respect is regarded as low. The current staff are generally very capable and are rated in this respect as very high, but the project is in a somewhat novel application domain for them so experience is rated as nominal. The toolsets available to the developers are judged to be typical for the size of company and are rated as nominal, as is the degree of schedule pressure to meet a deadline.

Given the data in Table 5.6.

(i) What would be the value for each of the effort multipliers?

(ii) What would be the impact of all the effort multipliers on a project estimated as taking 200 staff-months?

At a later stage of the project, detailed design of the application will have been completed. There will be a clearer idea of application size in terms of lines of code, and the factors influencing productivity will be better known. A revised estimate of effort can be produced based on the broader range of effort modifiers seen in Table 5.7. The method of calculation is the same as for early design. Readers who wish to apply the model using the post architecture effort multipliers are directed to the COCOMO II Model Definition Manual which is available from the University of Southern California website http://sunset.usc.edu/csse/research/COCOMOII/COCOMO_main.html.

5.14 Cost Estimation

Project cost can be obtained by multiplying the estimated effort (in man-month, from the effort estimate) with the manpower cost per month. Implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. However, in addition to manpower cost, a project would incur several other types of costs which we shall refer to as the overhead costs. The overhead costs would include the costs of hardware and software required for the project and the company overheads for administration, office space, etc. Depending on the expected values of the overhead costs, the project manager has to suitably scale up the cost estimated by using the COCOMO formula.

EXERCISE 5.14

Assume that the size of an organic type software product is estimated to be 32,000 lines of source code. Assume that the average salary of a software developer is £2,000 per month. Determine the effort required to develop the software product, the nominal development time, and the staff cost to develop the product.

5.15 Staffing Pattern

After the effort required to complete a software project has been estimated, the staffing requirement for the project can be determined. Putnam was the first to study the problem of what should be a proper staffing pattern for software projects. He extended the classical work of Norden who had earlier investigated the staffing pattern of general research and development (R&D) type of projects. In order to appreciate the staffing pattern desirable for software projects, we must understand both Norden's and Putnam's results.

Norden's work

Norden studied the staffing patterns of several R&D projects. He found the staffing patterns of R&D projects to be very different from that of manufacturing or sales type of work. In a sales outlet, the number of sales staff does not usually vary with time. For example, in a supermarket the number of sales personnel would depend on the number of sales counters alone and the number of sales personnel therefore remains fixed for years together. However, the staffing pattern of R&D type of projects changes dynamically over time for efficient manpower utilization. At the start of an R&D project, the activities of the project are planned and initial investigations are made. During this time, the manpower requirements are low. As the project progresses, the manpower requirement increases until it reaches a peak. Thereafter the manpower requirement gradually diminishes. Norden concluded that the staffing pattern for any R&D project can be approximated by the Rayleigh distribution curve shown in Figure 5.3.

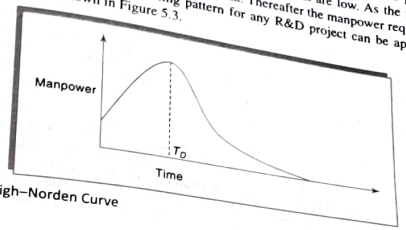


FIGURE 5.3 Rayleigh–Norden Curve

Putnam's work

Norden's work was carried out in the context of general R&D projects. Putnam studied the problem of staffing of software projects and found that the staffing pattern for software development projects has characteristics very similar to R&D projects. Putnam adapted the Rayleigh–Norden curve to relate the number of delivered lines of code to the effort and the time required to develop the product. Only a small number of developers are needed at the beginning of a project to carry out the planning and specification tasks. As the project progresses and more detailed work is performed, the number of developers increases and reaches a peak during product delivery which has been shown to occur at time T_D in Figure 5.3. After product delivery, the number of project staff falls consistently during product maintenance.

Putnam suggested that starting from a small number of developers, there should be a staff build-up and after a peak size has been achieved, staff reduction is required. However, the staff build-up should not be carried out in large instalments. Experience shows that a very rapid build-up of project staff any time during the project development correlates with schedule slippage.

EXERCISE 5.15

Suppose you are the project manager of a large development project. The top management informs that you would have to manage the project with a fixed team size throughout the duration of your project. What would be the likely impact of this decision on your project?

5.16 Effect of Schedule Compression

It is quite common for a project manager to encounter client requests to deliver products faster, that is, to compress the delivery schedule. It is therefore important to understand the impact of schedule compression on project cost. Putnam studied the effect of schedule compression on the development effort and expressed it in the form of the following equation:

$$pm_{new} = pm_{org} \times \left(\frac{td_{org}}{td_{new}}\right)^4$$

where pm_{new} is the new effort, pm_{org} is the originally estimated effort and td_{org} is the originally estimated time for project completion and td_{new} is the compressed schedule.

From this expression, it can easily be observed that when the schedule of a project is compressed, the required effort increases in proportion to the fourth power of the degree of compression. It means that a relatively small compression in a delivery schedule can result in substantial penalty on human effort. For example, if the estimated development time using COCOMO formula is one year, then in order to develop the product in six months, the total effort required (and hence the project cost) increases 16 times.

Boehm arrived at the result that there is a limit beyond which a software project cannot reduce its schedule by buying any more personnel or equipment. This limit occurs roughly at 75% of the nominal time estimate for small and medium sized projects. Thus, if a project manager accepts a customer demand to compress the development schedule of a typical project (medium or small project) by more than 25%, he is very unlikely to succeed. The main reason being, that every project has only a limited amount of activities which can be carried out in parallel, and the sequential activities cannot be speeded up by hiring any number of additional developers.

EXERCISE 5.16

The nominal effort and duration of a project is estimated to be 1000 pm and 15 months. The project cost is negotiated to be £200,000. This needs the product to be developed and delivered in 12 months time. What is the new cost that needs to be negotiated?

EXERCISE 5.17

Why does the effort requirement then increase drastically upon schedule compression (as per Putnam's results 16 times for schedule is compressed by 50%)? After all, isn't it the same product that is being developed?

5.17 Capers Jones Estimating Rules of Thumb

Capers Jones published a set of empirical rules in 1996 in the IEEE Computer journal. He formulated these rules based on his experience in estimating various parameters of a large number of software projects. Jones wanted that his rules should be as easy to use as possible, and yet should give the project manager a fair good idea of various aspects of a project. Because of their simplicity, these rules are handy to use for making off-hand estimates. However, these rules should not be expected to yield very accurate estimations and are certainly not considered appropriate for working out formal cost contracts. Still, while working out formal contracts, these rules are used to carry out sanity checks for estimations arrived using other more rigorous techniques. An interesting aspect of Jones' rules is that these rules give an insight into many aspects of a project (such as the rate of requirements creep) for which no formal methodologies exist as yet. In the following section, we discuss a few of Jones' rules of thumb that are often useful.

Rule 1: *SLOC Function Point Equivalence* One function point = 125 SLOC for C programs.

We have already pointed out in Section 5.4 that the SLOC measure is intuitive and helps in developing a good understanding of the size of a project. SLOC is also used in several popular techniques for estimating several project parameters. However, often the size estimations for a software project are done using the function point analysis due to the inherent advantages of the function point metric. In this situation, it often becomes necessary for the project manager to come up with the SLOC measure for the project from its function point measurements. Jones determined the equivalence between SLOC and function point for several programming languages based on experimental data.

To gain an insight into why SLOC function point equivalence varies across different programming languages let us examine the following. According to Jones, it would take about 320 lines of assembly code to implement one function point. Why does assembly coding take as much as three times the number of instructions required in C language to code one function point? It can be argued that to express one SLOC of C would require several instructions in assembly language.

Rule 2: *Project Duration Estimation* Function points raised to the power 0.4 predicts the approximate development time in calendar months.

To illustrate the applicability of this rule, consider that the size of a project is estimated to be 150 function points (that is, approximately 18,750 SLOC by Rule 1). The development time (time necessary to complete the project) would be about eight months by Rule 2.

Rule 3: *Rate of Requirements Creep* User requirements creep in at an average rate of 2% per month from the design through coding phases.

In almost every project, the features required by the customer keep on increasing due to a variety of reasons. Of course, requirement creeps are normally not expected during project testing and installation stages. Observe that the rule has been carefully worded to take into account the fact that while predicting the total requirements creep, it is necessary to remember that the requirement creeps occur from the end of the requirements phase till the testing phase. Therefore, only an appropriate fraction of the project completion period needs to be considered to exclude the durations of the requirements and testing phases.

Assume that the size of a project is estimated to be 150 function points. Then, the duration for this project can be estimated to be eight months by Jones' Rule 2. Since we need to exclude the duration of requirements specification and testing phase, it is reasonable to assume that the requirements creep would occur for five months only. By Rule 3, the original requirements will grow by a rate of three function points per month. So, the total requirements creep would roughly be 15 function points. Thus, the total size of the project for which the project manager needs to plan would be 165 function points rather than 150 function points.

Rule 4: *Defect Removal Efficiency* Each software review, inspection, or test step will find and remove 30% of the bugs that are present.

This rule succinctly captures the reason why software development organizations use a series of defect removal steps, viz., requirements review, design review, code inspection, and code walk-through, followed by unit, integration, and system testing. In fact, a series of about ten consecutive defect removal operations must be utilized to achieve good product reliability.

EXERCISE 5.18

For a certain project, consider that in the design document 1000 defects are present at the end of the design stage. Compute how many of these defects would survive after the processes of code review, unit, integration, and system testing have been completed. Assume that the defect removal effectiveness of each error removal stage is 30%.

Rule 5: *Project Manpower Estimation* The size of the software (in function points) divided by 150 predicts the approximate number of the personnel required for developing the application.

To understand the use of this rule, consider a project whose size is estimated to be 500 function points. By Rule 5, the number of development personnel required would be four. Observe that Rule 5 predicts the manpower requirement without considering several other relevant aspects of a project that can significantly affect the required effort. These aspects include the project complexity, the level of usage of CASE tools, and the programming language being used. It is therefore natural to expect that the actual manpower requirement would defer from that predicted by Rule 5. This inaccuracy, however, is in keeping with Jones' objective of having rules that are as simple as possible, so that these can be used off-hand to get a gross understanding of the important project parameters.

Rule 6: *Software Development Effort Estimation* The approximate number of staff months of effort required to develop a software is given by the software development time multiplied with the number of personnel required.

It can be observed that this rule is actually a corollary of the Rules 2 and 5. As an example for application of this rule, consider a project whose size is estimated to be 150 function points. Using Rules 2 and 5, the estimated development effort would be 8 months × 1 person = 8 person-months.

Rule 7: *Function points divided by 500 predicts the approximate number of personnel required for regular maintenance activities.*

According to Rule 1, 500 function points is equivalent to about 62,500 SLOC for C programs. Thus, we can say that approximately for every 60,000 SLOC, one maintenance personnel would be required to carry out minor bug fixes and functionality adaptations during the operation phase of the software.

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.

FURTHER EXERCISES

1. The size (that is, the effort needed to complete it) of any task will depend on its characteristics. The units into which the work is divided will also differ. Identify the factors affecting the size of the task and work units 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.
2. 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?
3. A small application maintains a telephone directory. The database for the application contains the following data types:
   - Staff reference
   - Surname
   - Forenames
   - Title
   - Department code
   - Room number
   - Telephone extension
   - E-mail address
   - Fax number

   Transactions are needed which:

   (i) set up new entries;
   (ii) mend existing entries;
   (iii) delete entries;
   (iv) allow enquirers to list online the details for a particular member of staff;
   (v) produce a complete listing of the telephone directory entries in alphabetical order.

   (a) Use this scenario to produce an estimated Mark II FP count. List all the assumptions you will need to make.

   (b) Another requirement could be to produce the listing in (v) in departmental order. In your view, should this increase FP count and if so by how much?

4. The following details are held about previously developed software modules.

Module	Inputs	Entity types accessed	Outputs	Days
a	1	2	10	2.60
b	10	1	1	3.90
c	5	3	11	1.83
d	2	3	1	3.50
e	1		20	4.30

A new module has 7 inputs, 1 entity type access and 7 outputs. Which of the modules a to e is the closest analogy in terms of Euclidean distance?

5. Using the data in further exercise 4 above, calculate the Symons Mark II FPs for each module. Using the results, calculate the effort needed for the new module described in further exercise 4. How does this estimate compare to the one based on analogy?

6. Given the project data below:

Project	Inputs	Outputs	Entity accesses	System users	Programming language	Developer days
1	210	420	40	10	x	30
2	469	1406	125	20	x	85
3	513	1283	76	18	y	108
4	660	2310	88	200	y	161
5	183	367	35	10	z	22
6	244	975	65	25	z	42
7	1600	3200	237	25	y	308
8	582	874	111	5	z	62
X	180	350	40	20	y	
Y	484	1190	69	35	y	

(a) What items are size drivers?

(b) What items are productivity drivers?

(c) What are the productivity rates for programming languages x, y and z?

(d) What would be the estimated effort for projects X and Y using a Mark II function point count

(e) What would be the estimated effort for X and Y using an approximate analogy approach?

(f) What would have been the best estimating method if the actual effort for X turns out to be 30 days and for Y turns out to be 120 days? Can you suggest why the results are as they are and how they might be improved?

7. A report in a college timetabling system produces a report showing the students who should be attending each timetabled teaching activity. Four files are accessed: the STAFF file, the STUDENT file, the STUDENT-OPTION file and the TEACHING-ACTIVITY file. The report contains the following information:

Teaching activity reference

Topic name

Staff forenames

Staff surname

Title

Semester (1 or 2)

Day of week

Time

Duration

Location

For each student:

student forename

student surnames

Calculate the Mark II FPs that this transaction would generate. What further information would you need to create an estimate of effort?

8. Suppose you are the manager of a software project. Explain why it would not be proper to calculate the number of developers required for the project as a simple division of the effort estimate (in person-months) by the nominal duration estimate (in months).

9. Suppose that off-the-shelf price of a certain management information system (MIS) software product is £50,000 and its size is 100 kdsi. Assuming that in-house developers cost £2000 per programmer-month (including overheads); would it be more cost-effective to buy the product or build it? Which elements of the cost are not included in COCOMO estimation model? What additional factors should be considered while making the decision to buy or build the product?

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 proje as a whole and for individual activities. A detailed plan for the project, however, must also include a schedu 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.

Project monitoring is discussed in more detail in Chapter 9.

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

project activity produces some tangible product or '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? In Chapter 5 we looked at ways of estimating the effort for various project tasks. However, 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. When should the resources 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.

Chapter 11 discusses motivation in more detail.

- *Coordination* 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 coordination among teams. In situations where staff may need to be transferred between project teams (or work concurrently on more 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.

This coordination will normally form part of Programme Management.

Activity planning and scheduling techniques place an emphasis on completing the project in a minimum time at an acceptable cost or, alternatively, meeting a 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 sched-uling 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. If we try relaxing precedence constraints by, for example, allowing a program coding task to commence before the design has been completed, it is up to us to ensure that we are clear about the potential effects on product quality.

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

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.

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).

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.

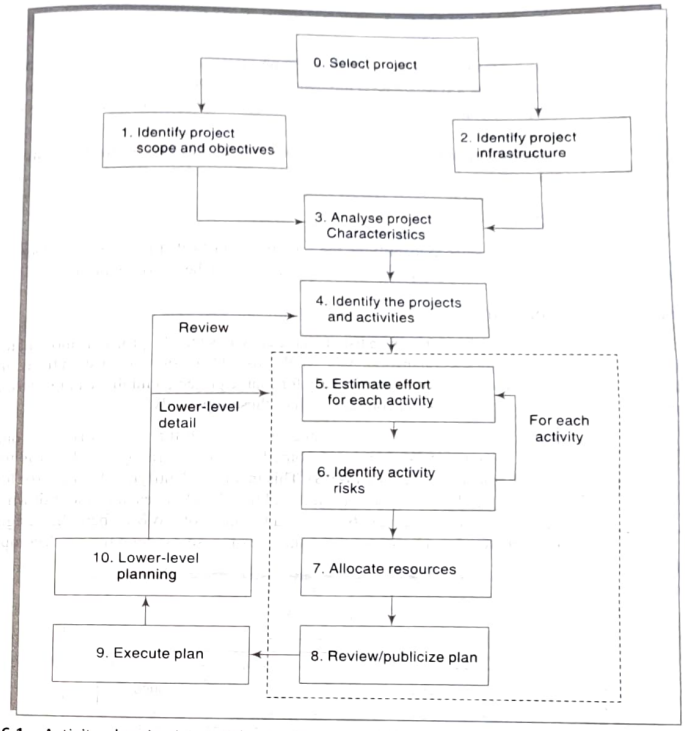


FIGURE 6.1 Activity planning is carried out in Steps 4 and 5

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.

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

- A project is composed of a number of interrelated 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*).

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 require 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-cycle stages and consider each of these separately.

WBSs are advocated by BS 6079, the British Standards Institution's *Guide to Project Management*.

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.

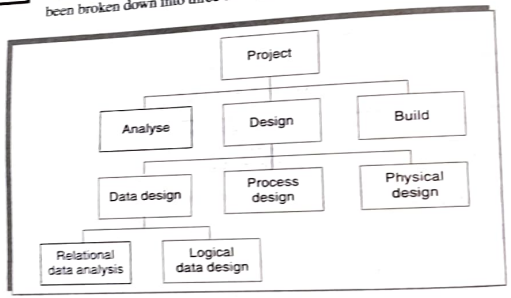


FIGURE 6.2 A fragment of an activity-based Work Breakdown Structure

Activities are added to a branch in the structure if they contribute directly 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 include everything that is required to complete the task at the higher level.

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 in the project – higher-level nodes merely represent collections of activities.

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

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 PRINCE2 and Step Wise, has already been described in Chapter 3. 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.

This approach is particularly appropriate if using a methodology such as SSADM or USDP (Unified Software Development Process), which clearly specifies, for each step or task, each of the products required and the activities required to produce it. For example, the SSADM Reference Manual provides a set of generic PBSs for each stage in SSADM, which can be used as a basis for generating a project specific PBS.

In the USDP, products are referred to as *artifacts* – see Figure 6.3 – and the sequence of activities needed to create them is called a *workflow* – see Figure 6.4 for an example. Some caution is needed in drawing up an activity network from these workflows. USDP emphasizes that processes are iterative. This means that it may not be possible to map a USDP process directly onto a single activity in a network. In Section 4.18 we saw how one or more iterated processes could be hidden in the single execution of a larger activity. All projects, whether they contain iterations or not, will need to have some fixed milestones or time-boxes if progress towards a planned delivery date is to be maintained. These larger activities with the fixed completion dates would be the basis of the activity network.

See I. Jacobson, G. Booch and J. Rumbaugh (1999) *The Unified Software Development Process*. Addison-Wesley.

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,

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

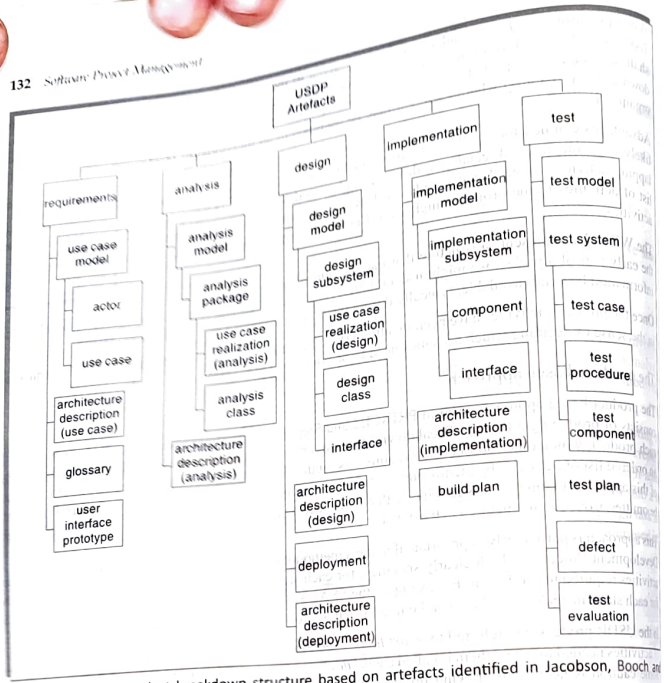


FIGURE 6.3 USDP product breakdown structure based on artefacts identified in Jacobson, Booch and Rumbaugh (1999)

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.

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

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.
- *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.

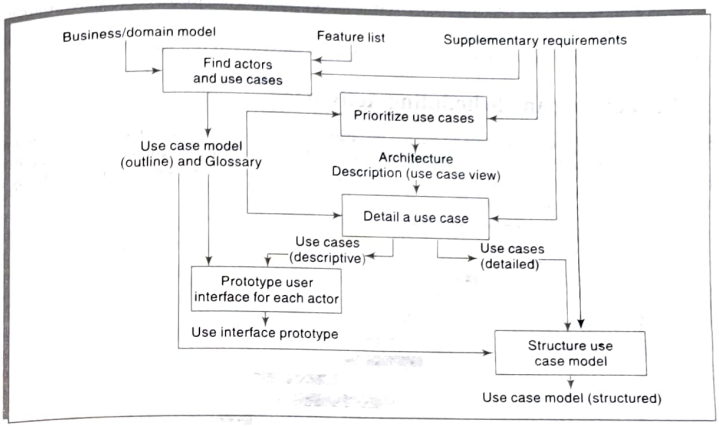


FIGURE 6.4 A structuring of activities for the USDP requirements capture workflow based on Jacobson, Booch and Rumbaugh (1999)

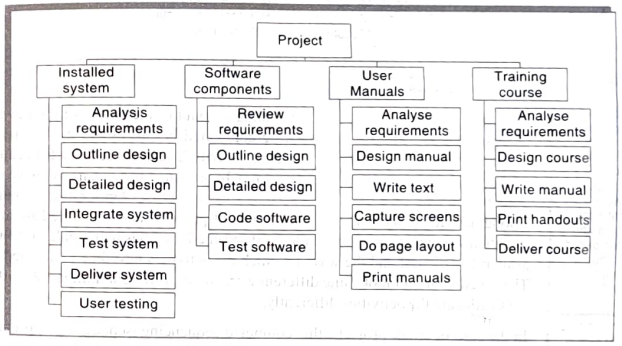


FIGURE 6.5 A hybrid Work Breakdown Structure based on deliverables and activities

- *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.

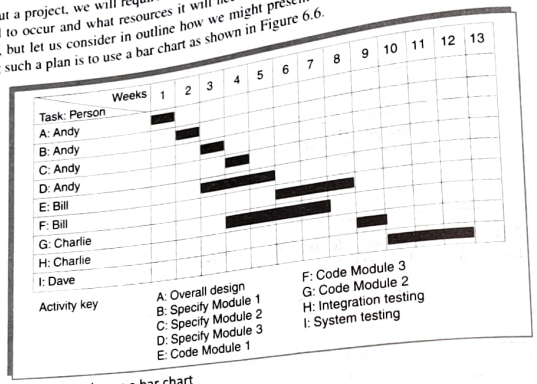


FIGURE 6.6 A project plan as a bar chart

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.

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 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.

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

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 tasks 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

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).

Both of these techniques used an *activity-on-arrow* approach to visualizing the project as a network where activities are drawn as arrows joining circles, or nodes, which represent the possible start and/or completion of an activity or set of activities. More recently a variation on these techniques, called *precedence networks*, has become popular. This method uses *activity-on-node* networks where activities are represented as nodes and the links between nodes represent precedence (or sequencing) requirements. This latter approach avoids some of the problems inherent in the activity-on-arrow representation and provides more scope for easily representing certain situations. It is this method that is adopted in the majority of computer applications currently available. These 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.

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

In the following sections of this chapter, we will look at the critical path method applied to precedence (activity-on-node) networks followed by a brief introduction to activity-on-arrow networks – a discussion of PERT will be reserved for Chapter 7 when we look at risk analysis.

CASE STUDY EXAMPLES

In Chapter 2 we saw how Amanda identified that three new software components would need to be developed and a further component would need to be rewritten. Figure 6.7 shows the fragment of a network that she has developed as an activity-on-node network. Figure 6.8 shows how this network would look represented as an activity-on-arrow network. Study each of the networks briefly to verify that they are, indeed, merely different graphical representations of the same thing.

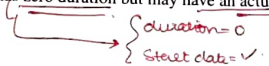
6.8 Formulating a Network Model

The first stage in creating a network model is to represent the activities and their interrelationships as a graph. In activity-on-node we do this by representing activities as nodes (boxes) in the graph – the lines between nodes represent dependencies.

Constructing precedence networks

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

A project network should have only one start node. Although it is logically possible to draw a network with more than one starting node, it is undesirable to do so as it is a potential source of confusion. In such cases (for example, where more than one activity can start immediately the project starts) it is normal to invent a 'start' activity which has zero duration but may have an actual start date.



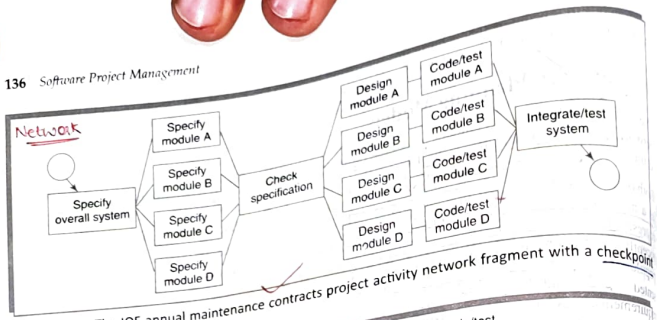


FIGURE 6.7 The IOE annual maintenance contracts project activity network fragment with a checkpoint activity added

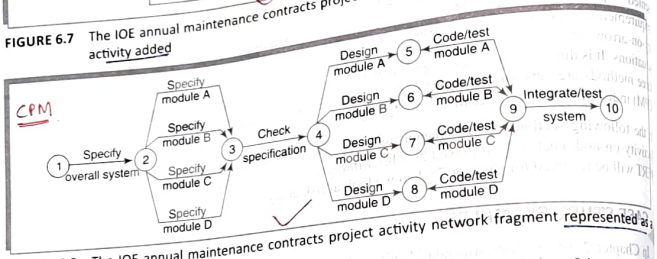


FIGURE 6.8 The IOE annual maintenance contracts project activity network fragment represented as a CPM network

(2) *A project network should have only one end node.* The end node designates the completion of the project and a project may finish only once! Although it is possible to draw a network with more than one end node, it will almost certainly lead to confusion if this is done. Where the completion of a project depends upon more than one 'final' activity it is normal to invent a 'finish' activity.

(3) *A node has duration.* A node represents an activity and, in general, activities take time to execute. Notice, however, that the network in Figure 6.7 does not contain any reference to durations. This network drawing merely represents the logic of the project – the rules governing the order in which activities are to be carried out.

Later we will look at the possibility of 'lag' between activities.

(4) *Links normally have no duration.* Links represent the relationships between activities. In Figure 6.9 installation cannot start until program testing is complete. Program testing cannot start until both coding and data take-on have been completed.

(5) *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 'Instal' 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 'Instal'. Note that we do not speak of 'Code' and 'Data take-on' as precedents of 'Instal' – that relationship is implicit in the previous statement.

(6) *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 some people add arrows to the lines to give a stronger visual indication of the time flow of the project.

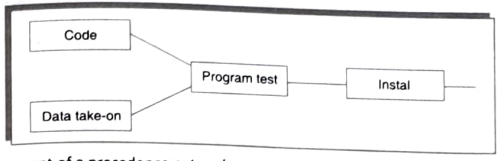


FIGURE 6.9 Fragment of a precedence network

(1) *A network may not contain loops.* Figure 6.10 demonstrates a loop in a 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.

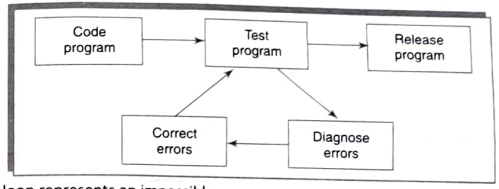


FIGURE 6.10 A loop represents an impossible sequence

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.

Although it is easy to see the loop in this simple network fragment, very large networks can easily contain complex loops which are difficult to spot when they are initially constructed. Fortunately, all network planning applications will detect loops and generate error messages when they are found.

(8) *A network should not contain dangles.* A dangling activity such as 'Write user manual' in Figure 6.11 should not exist as it is likely to lead to errors in subsequent analysis. Indeed, in many cases dangling activities indicate errors in logic when activities are added as an afterthought. If, in Figure 6.11, we mean to indicate that the project is complete once the software has been installed *and* the user manual written then we should

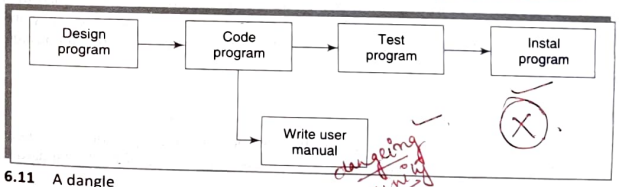


FIGURE 6.11 A dangle

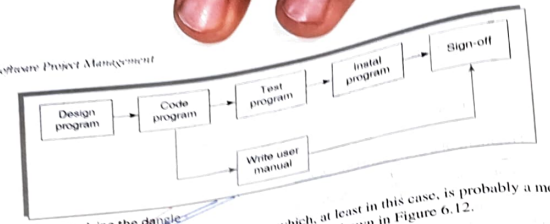


FIGURE 6.12 Resolving the dangle

redraw the network with a final completion activity – which, at least in this case, is probably a more accurate representation of what should happen. The redrawn network is shown in Figure 6.12.

Representing lagged activities

We might come across situations where we wish 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 is being tested – particularly when evaluating a prototype. In such a case we could designate an activity 'test and document amendments'. This would, however, make it impossible to show that amendment recording could start, say, one day after testing had begun and finish a little after the completion of testing.

Where activities can occur in parallel with a time lag between them, we represent the lag with a duration on the linking arrow as shown in Figure 6.13. This indicates that documenting amendments can start one day after the start of prototype testing and will be completed two days after prototype testing is completed.

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

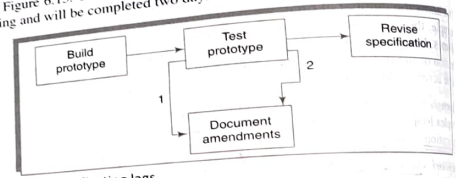


FIGURE 6.13 Indicating lags

Hammock activities

Hammock activities are activities which, in themselves, have zero duration but are assumed to start at the same time as the first 'hammocked' activity and to end at the same time as the last one. They are normally used for representing overhead costs or other resources that will be incurred or used at a constant rate over the duration of a set of activities.

Labelling conventions

There are a number of differing conventions that have been adopted for entering information on an activity-on-node network. The one adopted here is shown on the left and is based on the British Standard BS 4

The activity label is usually a code developed to uniquely identify the activity and may incorporate a project code (for example, IoE/P/3 to designate one of the programming activities for IOE's annual maintenance contract project). The activity description will normally be a brief activity name such as 'Test take-on module'. The other items in our activity node will be explained as we discuss the analysis of a project network.

Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish

6.9 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 approach 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. Brigette at Brightmouth College has completed the software package evaluation and a software package has been chosen and approved. Now that the application software is known, the hardware needed as a platform can be acquired. Another task will be 'system configuration' – there are a number of parameters that will have to set in the application so that it runs satisfactorily for Brightmouth College. Once the parameters have been set, details of the employees who are to be paid will have to be set up on the new system. Enough information about the new system will now be available so that office procedures can be devised and documented. There are currently no staff currently dedicated to payroll administration so a payroll officer is to be recruited and then trained.

EXERCISE 6.1

Draw an activity network using precedence network conventions for the project specified in Table 6.1. When you have completed it, compare your result with that shown in Figure 6.14.

Figure 6.14 illustrates the network for the project specified in Table 6.1.

6.10 The Forward Pass

The forward pass is carried out to calculate the earliest dates on which each activity may be started and completed.

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

	Activity	Duration (weeks)	Precedents
A	Hardware selection	6	
B	System configuration	4	
C	Instal hardware	3	A
D	Data migration	4	B
E	Draft office procedures	3	B
F	Recruit staff	10	
G	User training	3	E, F
H	Instal and test system	2	C, D

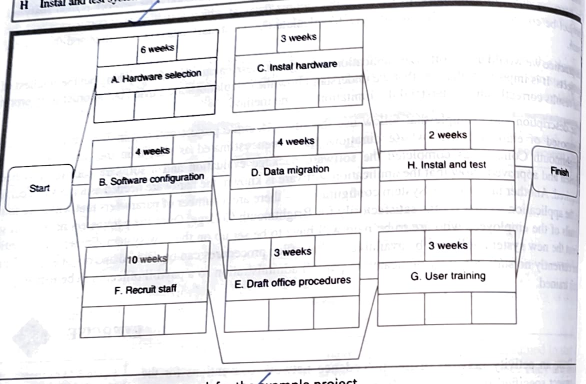


FIGURE 6.14 The precedence network for the example project

Where an actual start date is known, the calculations may be carried out using actual dates. Alternatively we can use day or week numbers and that is the approach we shall adopt here. By convention, dates indicate the end of a period and the project is therefore shown as starting at the end of week zero (or the beginning of week 1).

The forward pass and the calculation of earliest start dates are carried out according to the following reasoning.

- Activities A, B and F may start immediately, so the earliest date for their start is zero.
- Activity A will take 6 weeks, so the earliest it can finish is week 6.
- Activity B will take 4 weeks, so the earliest it can finish is week 4.
- Activity F will take 10 weeks, so the earliest it can finish is week 10.
- Activity C can start as soon as A has finished so its earliest start date is week 6. It will take 3 weeks so the earliest it can finish is week 9.
- Activities D and E can start as soon as B is complete so the earliest they can each start is week 4. Activity D, which will take 4 weeks, can therefore finish by week 8 and activity E, which will take 3 weeks, can therefore finish by week 7.
- Activity G cannot start until both E and F have been completed. It cannot therefore start until week 10 – the later of weeks 7 (for activity E) and 10 (for activity F). It takes 3 weeks and finishes in week 13.
- Similarly, Activity H cannot start until week 9 – the later of the two earliest finish dates for the preceding activities C and D.
- The project will be complete when both activities H and G have been completed. Thus the earliest project completion date will be the later of weeks 11 and 13 – that is, week 13.

During the forward pass, earliest dates are recorded as they are calculated.

The forward pass rule: the earliest start date for an activity is the earliest finish date for the preceding activity. Where there is more than one immediately preceding activity we take the latest of the earliest finish dates for those activities.

The results of the forward pass are shown in Figure 6.15.

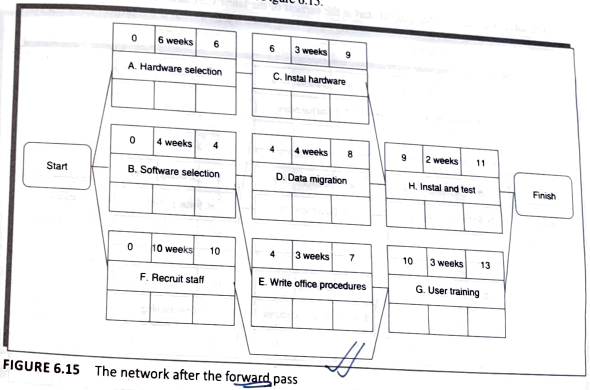


FIGURE 6.15 The network after the forward pass

6.11 The Backward Pass

The backward pass rule: the latest finish date for an activity is the latest start date for the activity that commences immediately that activity is complete. Where more than one activity can commence we take the earliest of the latest start dates for those activities.

The second stage in the analysis of a critical path network is to carry out a backward pass to calculate the latest date at which each activity may be started and finished without delaying the end date of the project. 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 6.16 illustrates our network after carrying out the backward pass. The latest activity dates are calculated as follows.

- The latest completion date for activities G and H is assumed to be week 13.
- Activity H must therefore start at week 11 at the latest (13 – 2) and the latest start date for activity G is week 10 (13 – 3).
- The latest completion date for activities C and D is the latest date at which activity H must start – that is, week 11. They therefore have latest start dates of week 8 (11 – 3) and week 7 (11 – 4) respectively.
- Activities E and F must be completed by week 10 so their earliest start dates are weeks 7 (10 – 3) and 0 (10 – 10) respectively.
- Activity B must be completed by week 7 (the latest start date for both activities D and E) so its latest start is week 3 (7 – 4).
- Activity A must be completed by week 8 (the latest start date for activity C) so its latest start is week 2 (8 – 6).
- The latest start date for the project start is the earliest of the latest start dates for activities A, B and F. This is week 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.

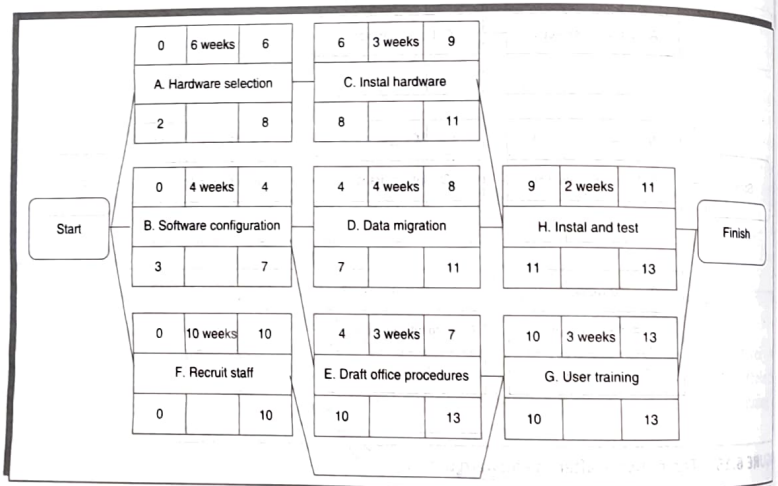


FIGURE 6.16 The network after the backward pass

6.12 Identifying the Critical Path

There will be at least one path through the network (that is, one set of successive activities) that defines the duration of the project. This is known as the *critical path*. Any delay to any activity on this critical path will delay the completion of the project.

The critical path is the longest path through the network.

The difference between an activity's earliest start date and its latest start date (or, equally, the difference between its earliest and latest finish dates) is known as the activity's *float* – it is a measure of how much the start or completion of an activity may be delayed without affecting the end date of the project. Any activity with a float of zero is critical in the sense that any delay in carrying out the activity 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 activities – this path is known as the critical path and is shown bold in Figure 6.17.

This float is also known as total float to distinguish it from other forms of float – see Section 6.13.

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.

Figure 6.17 also shows the *activity span*. This 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. However, it is subject to the same conditions of interpretation as activity float, which is discussed in the next section.

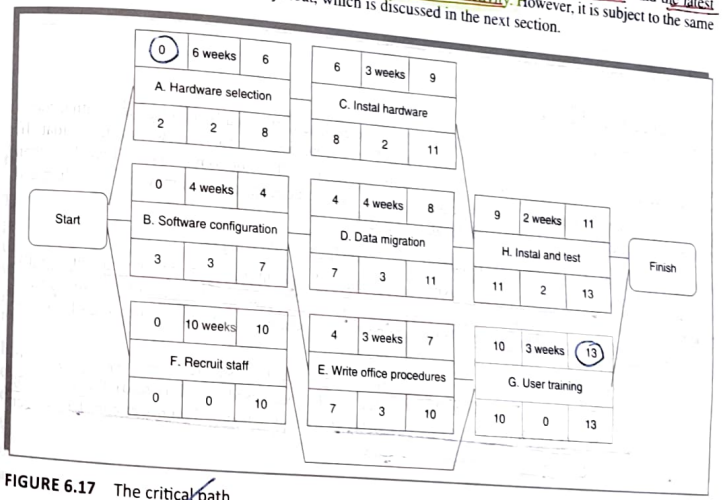


FIGURE 6.17 The critical path

EXERCISE 6.2

Refer back to Amanda's CPM network illustrated in Figure 6.7.

Using the activity durations given in Table 6.2, calculate the earliest completion date for the project and identify the critical path on your network.

TABLE 6.2 Estimated activity durations for Amanda's network

Activity	Estimated duration (days)	Activity	Estimated duration (days)
Specify overall system	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		

6.13 Activity Float

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 in Figure 6.17 each have 2 weeks' total float but, however, activity A uses up its float (that is, it is not completed until week 8) the 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. This 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 – although it will delay the start of subsequent activities.

EXERCISE 6.3

Calculate the free float and interfering float for each of the activities shown in the activity network (Figure 6.17).

6.14 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 6.17, 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).

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).

EXERCISE 6.4

Referring to Figure 6.17, 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?

6.15 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.

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

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.

6.16 Activity-on-Arrow Networks

The developers of the CPM and PERT methods both originally used activity-on-arrow networks. Although now less common than activity-on-node networks, they are still used and introduce an additional useful concept – that of events. We will therefore take a brief look at how they are drawn and analysed using the same project example shown in Table 6.1.

In activity-on-arrow networks activities are represented by links (or arrows) and the nodes represent events of activities (or groups of activities) starting or finishing. Figure 6.18 illustrates our previous example (see Figure 6.14) drawn as an activity-on-arrow network.

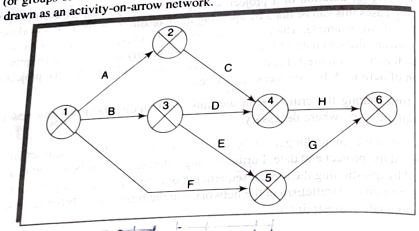


FIGURE 6.18 An activity-on-arrow network

Activity-on-arrow network rules and conventions

1. *A project network may have only one start node.* This is a requirement of activity-on-arrow networks rather than merely desirable as is the case with activity-on-node networks.
2. *A project network may have only one end node.* Again, this is a requirement for activity-on-arrow networks.
3. *A link has duration.* A link represents an activity and, in general, activities take time to execute. Note, however, that the network in Figure 6.18 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.

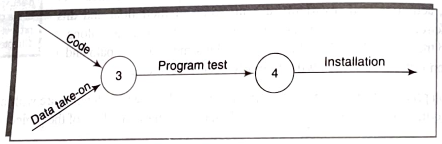


FIGURE 6.19 Fragment of a CPM network

4. *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 into 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 6.19, node 3 is the event that both 'coding' and 'data take-on' have been completed and activity 'program test' is free to start. 'Installation' may be started only when event 4 has been achieved, that is, as soon as 'program test' has been completed.

4. *Time moves from left to right.* As with activity-on-node networks, activity-on-arrow networks are drawn, if at all possible, so that time moves from left to right.
5. *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.
6. *A network may not contain loops.* Figure 6.20 demonstrates a loop in an activity-on-arrow network. As discussed in the context of precedence networks, loops are either an error of logic or a situation that must be resolved by itemizing iterations of activity groups.

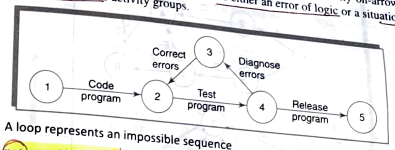


FIGURE 6.20 A loop represents an impossible sequence

6. *A network may not contain dangles.* A dangling activity, such as 'Write user manual' in Figure 6.21, cannot exist, as it would suggest there are two completion points for the project. If, in Figure 6.21, 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. 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.

Dangles are not allowed in activity-on-arrow networks.

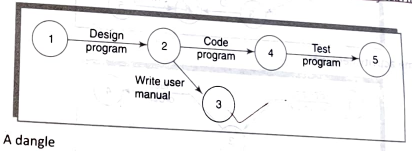


FIGURE 6.21 A dangle

EXERCISE 6.5

Take a look at the networks in Figure 6.22. State what is wrong with each of them and, where possible, redraw them correctly.

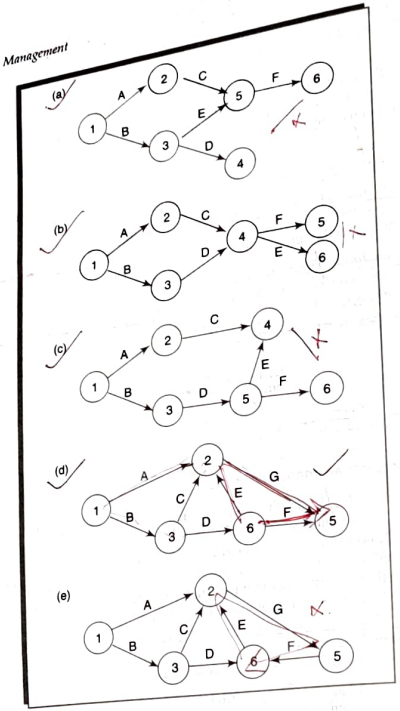


FIGURE 6.22 Some activity networks

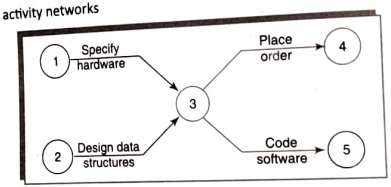


FIGURE 6.23 Two paths with a common node

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 6.23 might occur.

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 6.23 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 'specify hardware' to the start of the activity 'code software'. This effectively breaks the link between data structure design and placing the order and is shown in Figure 6.24.

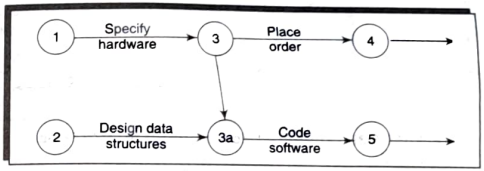


FIGURE 6.24 Two paths linked by a dummy activity

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 6.25. 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.

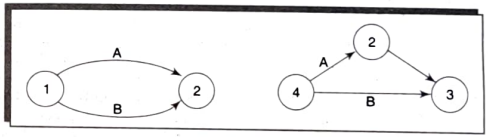


FIGURE 6.25 Another use of a dummy activity

These are problems that do not occur with activity-on-node networks.

EXERCISE 6.6

Take another look at Brigette's college payroll activity network fragment, which related to the earlier software selection process and which you developed in Exercise 3.4 (or take a look at the model answer in Figure B.2). Redraw this as an activity-on-arrow network.

Representing lagged activities

Activity-on-arrow networks are less elegant when it comes to representing lagged parallel activities. We need to represent these with pairs of dummy activities as shown in Figure 6.26. 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.

Activity labelling

There are a number of differing conventions that have been adopted for entering information on an activity-on-arrow 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.

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

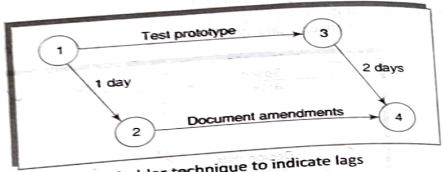
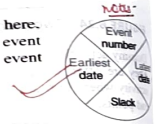


FIGURE 6.26 Using the ladder technique to indicate lags

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).



- 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).
- 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 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.

The results of the forward pass are shown in Figure 6.27 and Table 6.3.

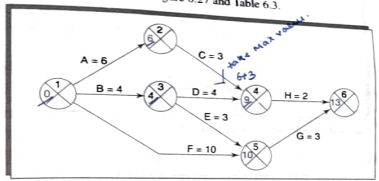


FIGURE 6.27 A CPM network after the forward pass

TABLE 6.3 The activity table after the forward pass

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 last date by which all immediately following activities must be started for the project to be completed on time. As with activity-on-node networks, we assume that the 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.

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 6.28 illustrates our network and Table 6.4 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.

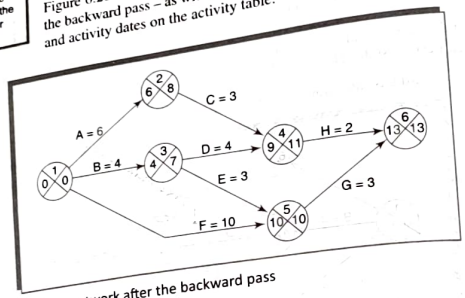
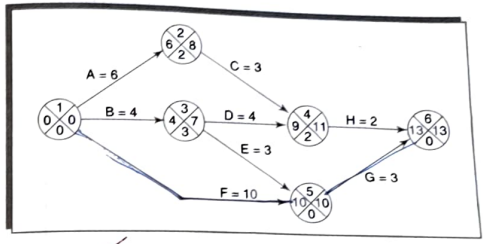


FIGURE 6.28 The CPM network after the backward pass

TABLE 6.4 The activity table following the backward pass

Activity	Duration (weeks)	Earliest start date	Latest start date	Earliest finish date	Latest finish date	Total
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	

Identifying the critical path The critical path is identified in a way similar to that used in activity-on-node networks. We do, however, use a different concept, that of *slack*, in identifying the path. Slack is the difference between the earliest date and the latest date for an event – it is a measure of how late an event may be without affecting the end date of the project. The critical path is the path joining all nodes with a zero slack (Figure 6.29).



The critical path is the longest path through the network.

FIGURE 6.29 The critical path

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.

FURTHER EXERCISES

1. Draw an activity network using either activity-on-node or activity-on-arrow network conventions for each of the following projects:
   - redecorating a room;
   - 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 annual maintenance contracts project. Base your plan on that used for Exercise 6.2 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 6.2, discuss what options Amanda might consider if she found it necessary to complete the project earlier than day 104.
4. Create a precedence activity network using the following details:

Activity	Depends on	Duration (days)
A		5
B		7
C	A	6
D	B	5
E	A	10
F	D	15
G	B	8
H	B	8
I	G	4
J	C	4
K	G	5
L	E, F	3
	I, H	

5. Calculate the earliest and latest start and end dates and the float associated with each activity in the network you have created for further exercise 4 above. From this identify the critical path.
6. Draw up a precedence activity network for the following scenario: The specification of an ICT application is estimated as likely to take two weeks to complete. When this activity has been complete work can start on three software modules, A, B and C. Design/coding of the modules will need 5, and 10 days respectively. Modules A and B can only be unit-tested together as their functionality closely associated. This joint testing should take about two weeks. Module C will need eight days unit testing. When all unit testing has been completed, integrated system testing will be needed, taki a further three weeks. This testing will be based on the functionality described in the specification will need 10 days of planning.
7. For the activity network in further exercise 6 above, derive the earliest and latest start dates for e activity and the earliest and latest finish dates. Work out the shortest project duration. If only software developers were available for the design and coding of modules, what effect would this h on the project duration?
8. What are the limitations of the precedence and CPM activity network notations?
9. Consider a software project with five tasks T1–T5. Duration of the five tasks in weeks is 3, 2, 3, 5, a 2 respectively. T2 and T4 can start when T1 is complete, T3 can start when T2 is complete. T5 can s when both T3 and T4 are complete. Draw the CPM network representation of the project. When is t latest start date of the task T3? What is the float time of the task T4? Which tasks are on the criti path?

7

RISK MANAGEMENT

OBJECTIVES

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

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

7.1 Introduction

In Chapter 6 we saw how, at IOE, Amanda planned how the software for the new annual maintenance contracts application was to be produced. This included estimating how long each task would take – see Figure 6.7 and Table 6.2. Her plan was based on the assumption that three experienced programmers were available for the coding of modules A, B, C and D. However, suppose two developers then left for better-paid jobs, and so far only one replacement has been recruited, who happens to be a trainee.

In the case of Brigette and the Brightmouth payroll implementation project, imagine that a payroll package has been purchased. However, a new requirement emerges that the payroll database should be accessed by a new application that calculates the staff costs for each course delivered by the college. Unfortunately, the purchased payroll application does not allow this access.

In some work environments 'problems' in this context are referred to as 'issues'.

Amanda and Brigette will have to deal with these *problems* as part of the monitoring and control process that will be outlined in Chapter 9. In this chapter we consider whether the two project leaders could have foreseen that these problems were likely to occur and made plans to deal with them. In other words, could these problems have been identified as *risks*?

7.2 Risk

PM-BOK stands for Project Management Body of Knowledge, a project management standard published by the Project Management Institute in the USA.

PM-BOK defines risk as *an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives*. PRINCE2, the UK government sponsored project management standard, defines risk as *the chance of exposure to adverse consequences of future events*. The two definitions differ, as the first includes situations where a future uncertainty actually works in our favour and presents us with an opportunity. We will return to this later in the chapter.

The key elements of a risk follow.

1. *It relates to the future.* The future is inherently uncertain. Some things which seem obvious when a project is over, for example that the costs were underestimated or that a new technology was difficult to use, might not have been so obvious during planning.
2. *It involves cause and effect.* For example, a 'cost over-run' might be identified as a risk, but 'cost over-run' describes some damage, but does not say why it occurred. Is it, for example, an inaccurate estimate of effort, the use of untrained staff or a poor specification? Both the cause (or *hazard*), such as 'inexperienced staff', and a particular type of negative outcome, such as 'lower productivity', should be defined for each risk.

The ISPL risk model (formerly Euromethod) refers to hazards as 'situational factors'.

EXERCISE 7.1

Match the following causes – a to d – to their possible effects – i to iv. The relationships are not necessarily one-to-one. Explain the reasons for each match.

Causes

(a) staff inexperience;
(b) lack of top management commitment;
(c) new technology;
(d) users uncertain of their requirements.

Effects

(i) testing takes longer than planned;
(ii) planned effort and time for activities exceeded;
(iii) project scope increases;
(iv) time delays in getting changes to plans agreed.

The boundary between risk management and 'normal' software project management is hazy. For when we were selecting the best general approach to a project – see Chapter 4 – one considerati possible consequences of future adverse events. As will be seen in Chapter 13, most of the techniq assure the quality of software, such as reviews and testing, are designed to reduce the risk of fault deliverables. Risk management is not a self-contained topic within project management. The risk management is considering uncertainty remaining after a plan has been formulated. Every pl on assumptions and risk management tries to plan for and control the situations where those a become incorrect. Risk planning is carried out in Steps 3 and 6 (Figure 7.1).

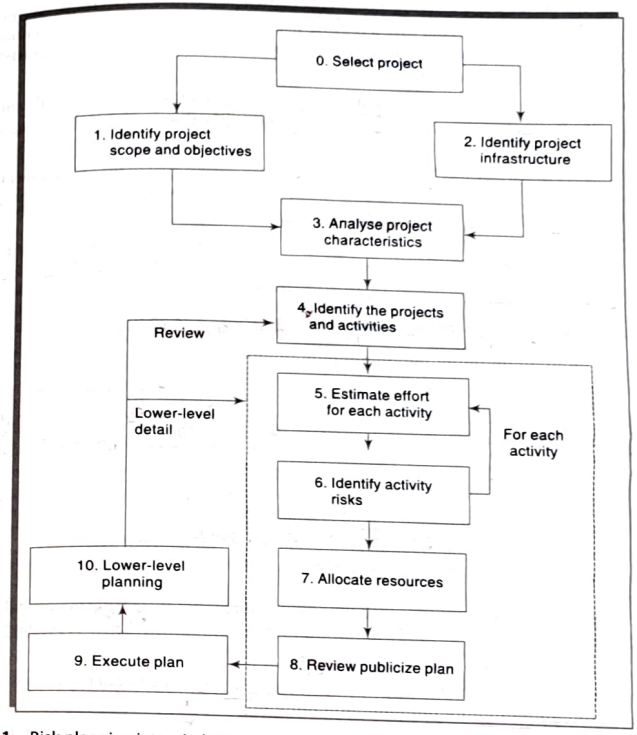


FIGURE 7.1 Risk planning is carried out primarily in Steps 3 and 6

7.3 Categories of Risk

An ICT project manager is normally given the objective of installing the required application by a specified deadline and within an agreed budget. Other objectives might be set, especially with regard to quality requirements. *Project risks* are those that could prevent the achievement of the objectives given to the project manager and the project team.

As we noted in Chapter 2, there could be risks that an application after successful implementation is a business failure. Thus if an e-commerce site is established to sell a product, the site might be correctly implemented, but customers fail to use the site because of the uncompetitive prices demanded. Dealing with these *business*

Risks is likely to be outside the direct responsibilities of the application implementation team. However, the failure to meet any project objective could have a negative impact on the business case for the project. For example, an increase in development cost might mean that the income (or savings) generated by the delivered application no longer represents a good return on the increased investment.

Risks have been categorized in other ways. Kalle Lyytinen and his colleagues, for instance, have proposed a sociotechnical model for risk, a diagrammatic representation of which appears in Figure 7.2.

See K. Lyytinen, L. Mathiassen and J. Ropponen (1996) 'A framework for risk management' *Journal of Information Technology*, 11(4).

The box labelled *Actors* refers to all the people involved in the development of the application in question. A typical risk in this area is that high staff turnover leads to expertise of value to the project being lost.

In Figure 7.2, the box labelled *Technology* encompasses both the technology used to implement the application and that embedded in the delivered products. Risks here could relate to the appropriateness of the technologies and to possible faults within them, especially if they are novel.

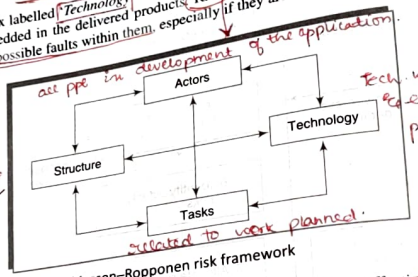


FIGURE 7.2 The Lyytinen–Mathiassen–Ropponen risk framework

Structure describes the management structures and systems, including those affecting planning and control. For example, the implementation might need user participation in some tasks, but the responsibility for managing the users' contribution might not be clearly allocated.

Tasks relates to the work planned. For instance, the complexity of the work might lead to delays because of the additional time required integrate the large number of components.

In Figure 7.2 all boxes are interlinked. Risks often arise from the relationships between factors – for example between technology and people. If a development technology is novel then the developers might not be experienced in its use and delay results. The novelty of the new technology is really a characteristic of the developers: once they are used to the technology, it is no longer 'novel'.

EXERCISE 7.2

In the cases of the Brightmouth payroll implementation project and the IOE annual maintenance contracts development project, identify one risk for each of the four categories in Figure 7.2.

7.4 A Framework for Dealing with Risk

Planning for risk includes these steps:

(i) risk identification;
(ii) risk analysis and prioritization;
(iii) risk planning;
(iv) risk monitoring.

Steps (i) to (iii) above will probably be repeated. When risks that could prevent a project success are identified, plans can be made to reduce or remove their threat. The plans are then reassessed to ensure that the original risks are reduced sufficiently and no new risks inadvertently introduced. Take the risk that staff inexperience with a new technology could lead to delays in software development. To reduce this risk, consultants expert in the new technology might be recruited. However, the use of consultants might introduce the new risk that knowledge about the new technology is not transferred to the permanent staff, making subsequent software maintenance problematic. Having identified this new risk, further risk reduction activities can be planned.

7.5 Risk Identification

The two main approaches to the identification of risks are the use of *checklists* and *brainstorming*.

Checklists are simply lists of the risks that have been found to occur regularly in software development projects. A specialized list of software development risks by Barry Boehm appears in Table 7.1 in a modified version. Ideally a group of representative project stakeholders examines a checklist identifying risks applicable to their project. Often the checklist suggests potential countermeasures for each risk.

TABLE 7.1 Software project risks and strategies for risk reduction

Risk	Risk reduction techniques
(1) Personnel shortfalls	Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel
(2) Unrealistic time and cost estimates	Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods
(3) Developing the wrong software functions	Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals
(4) Developing the wrong user interface	Prototyping; task analysis; user involvement
(5) Gold plating	Requirements scrubbing; prototyping; cost–benefit analysis; design to cost

(*Contd*)

This top ten list of software risks is based on one presented by Barry Boehm in his *Tutorial on Software Risk Management*, IEE Computer Society, 1989.

(Contd)

Late changes to requirements	Stringent change control procedures; high change threshold; incremental development (deferring changes)
Shortfalls in externally 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; 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

The 'lessons learnt' report differs from a 'post implementation review' (PIR). It is written on project completion and focuses on project issues. A PIR, produced when the application has been operational for some time, focuses on business benefits.

Project management methodologies, such PRINCE2, often recommend that on completion of a project a review identifies any problems during the project and the steps that were (or should have been) taken to resolve or avoid them. These problems could in some cases be added to an organizational risk checklist for use with new projects.

Brainstorming

Ideally, representatives of the main stakeholders should be brought together once some kind of preliminary plan has been drafted. They then identify, using their individual knowledge of different parts of the project, the problems that might occur. This collaborative approach may generate a sense of ownership in the project.

'Brainstorming' is also mentioned in Chapter 13 in connection with quality circles.

Brainstorming might be used with Brigette's Brightmouth payroll implementation project as she realizes that there are aspects of college administration of which she is unaware. She therefore suggests to the main stakeholders in the project, who include staff from the finance office and the personnel office, that they meet and discuss where the risks facing the project lie.

7.6 Risk Assessment

A common problem with risk identification is that a list of risks is potentially endless. A way is needed of distinguishing the damaging and likely risks. This can be done by estimating the *risk exposure* for each risk using the formula:

risk exposure = (potential damage) × (probability of occurrence)

Using the most rigorous – but not necessarily the most practical – approach, the potential damage would be assessed as a money value. Say a project depended on a data centre vulnerable to fire. It might be estimated that if a fire occurred a new computer configuration could be established for £500,000. It might also be estimated that where the computer is located there is a 1 in 1000 chance of a fire actually happening, that is a probability of 0.001.

The risk exposure in this case would be:

£500,000 × 0.001 = £500

A crude way of understanding this value is as the minimum sum an insurance company would require as a premium. If 1000 companies, all in the same position, each contributed £500 to a fund then, when the 1 in 1000 chance of the fire actually occurred, there would be enough money to cover the cost of recovery.

EXERCISE 7.3

What conditions would have to exist for the risk pooling arrangement described above to work?

The calculation of risk exposure above assumes that the amount of damage sustained will always be the same. However, it is usually the case that there could be varying amounts of damage. For example, as software development proceeds, more software is created, and more time would be needed to re-create it if it were lost.

With some risks, there could be not only damage but also gains. The testing of a software component is scheduled to take six days, but is actually done in three days. A team leader might therefore feel justified in producing a probability chart like the one in Figure 7.3. This shows the probability of a task being completed in four days (5%), then five days (10%), and so on. The accumulated probability for the seventh day (65%) means that there is a 65% chance that the task will be finished on or before the seventh day.

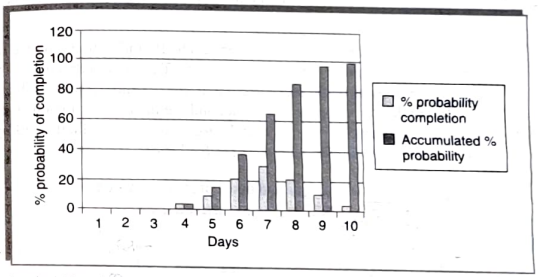


FIGURE 7.3 Probability chart

Clients would almost certainly insist we pick one of the days as the target. This target could be 'aggressive', for instance only five days in the above scenario, but with an 85% chance of failure according to the chart. A safer estimate would be eight days which would have a probability of failure of only 15%. We will return to this point later on in this chapter.

In Figure 7.3 the 'loss' is effectively being measured in days rather than money. In this context, days, or some other unit of personal effort, is often used as a *surrogate* for a financial loss.

Most managers resist very precise estimates of loss or of the probability of something occurring, as such figures are usually guesses. Barry Boehm has suggested that, because of this, both the risk losses and the

probabilities be assessed using relative scales in the range 0 to 10. The two figures could then be multiplied together to get a notional risk exposure. Table 7.2 provides an example, based on Amanda's IOE group accounts project, of where this has been done. This value could be used to prioritize the importance of risks although more sophisticated risk calculations are not possible.

TABLE 7.2 Part of Amanda's risk exposure assessment

Ref	Hazard	Likelihood	Impact	Risk
R1	Changes to requirements specification during coding	8	8	64
R2	Specification takes longer than expected	3	7	21
R3	Significant staff sickness affecting critical path activities	5	7	35
R4	Significant 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	4	8	32

Boehm suggests that planners focus attention on the 10 risks with the highest risk exposure scores. For smaller projects – including the final-year projects of computing students – the focus could be on a smaller number of risks.

See P. Goodwin and G. Wright (2004) *Decision Analysis for Management Judgement*, Wiley, for further discussion of this issue.

Even using indicative numbers in the range 0 to 10, rather than precise money values and probabilities, is not completely satisfactory. The values are likely to be subjective and different analysts might pick different numbers. Another approach is to use qualitative descriptions of the possible impact and the likelihood of each risk – see Tables 7.3 and 7.4 for examples. Consistency between assessors is facilitated by associating each qualitative description with a range of values.

TABLE 7.3 Qualitative descriptors of risk probability and associated range values

Probability level	Range
High	Greater than 50% chance of happening
Significant	30–50% chance of happening
Moderate	10–29% chance of happening
Low	Less than 10% chance of happening

TABLE 7.4 Qualitative descriptors of impact on cost and associated range values

Impact level	Range
High	More than 30% above budgeted expenditure
Significant	20 to 29% above budgeted expenditure
Moderate	10 to 19% above budgeted expenditure
Low	Within 10% of budgeted expenditure.

In Table 7.4, the potential amount of damage has been categorized in terms of its impact on *project costs*. Other tables could show the impact of risks on *project duration* or on the *quality of the project deliverables*.

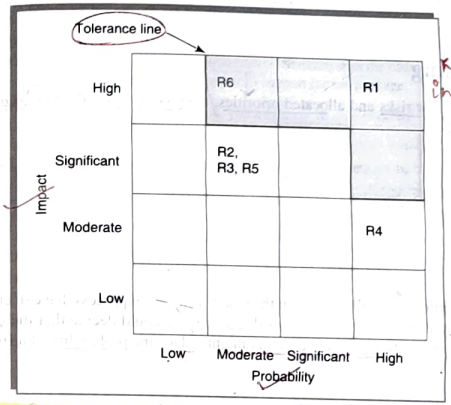


FIGURE 7.4 A probability impact matrix

To some extent, the project manager, in conjunction with the project sponsor, can choose whether the damage inflicted by a risk affects cost, duration or the quality of deliverables. In Amanda's list of risks in Table 7.2, R5 refers to the coding of modules taking longer than planned. This would have an impact on both the duration of the project and the costs, as more staff time would be needed. A response might be adding software developers and splitting the remaining development work between them. This will increase costs, but could save the planned completion date. Another option is to save both duration and staff costs by reducing software testing before the software is released. This is likely to be at the price of decreased quality in the project deliverable.

Where the potential damage and likelihood of a risk are defined by qualitative descriptors, the risk exposure cannot be calculated by multiplying the two factors together. In this case, the risk exposure is indicated by the

position of the risk in a matrix – see Figure 7.4. These matrices have variously been called *probability impact grids* or *summary risk profiles*.

In Figure 7.4 some of the cells in the top right of the matrix have been zoned off by a *tolerance line*. Risk that appear within this zone have a degree of seriousness that calls for particular attention.

The term *risk proximity* is used to describe this attribute of risk.

Chapter 5 stressed the need for frequent reassessment of effort and duration estimates during a project. This applies to risk exposure as well, as some risks apply only at certain stages. A risk might be that key users are unavailable when needed to supply details of their requirements. As requirements are gathered, so this risk will diminish. In general, the element of uncertainty will lessen as a project progresses and more is learnt by the developers about user requirements and any new technology. This would be reflected in lower risk probabilities. On the other hand, the potential damage will tend to increase as the amount invested in the project grows. If you type a substantial report using a word processor and neglect to take back-ups, each day adds more text to the report, it also adds to the number of days needed to re-key the report in the event of file loss.

7.7 Risk Planning

Having identified the major risks and allocated priorities, the task is to decide how to deal with them. The choices discussed will be:

- risk acceptance;
- risk avoidance;
- risk reduction and mitigation;
- risk transfer.

Risk acceptance

This is the do-nothing option. We will already, in the risk prioritization process, have decided to ignore some risks in order to concentrate on the more likely or damaging. We could decide that the damage inflicted by some risks would be less than the costs of action that might reduce the probability of a risk happening.

Risk avoidance

Some activities may be so prone to accident that it is best to avoid them altogether. If you are worried about sharks then don't go into the water. For example, given all the problems with developing software solutions from scratch, managers might decide to retain existing clerical methods, or to buy an off-the-shelf solution.

Risk reduction

It must be appreciated that each risk reduction action is likely to involve some cost. This is discussed in the next section.

Here we decide to go ahead with a course of action despite the risks, but take precautions that reduce the probability of the risk.

This chapter started with a scenario where two of the staff scheduled to work on Amanda's development project at IOE departed for other jobs. If this has been identified as a risk, steps might have been taken to reduce possible departures of staff.

For instance, the developers might have been promised generous bonuses to be paid on successful completion of the project.

Recall that Brigette had a problem at Brightmouth College: after the purchase of the payroll package, a requirement for the payroll database to be accessed by another application was identified. Unfortunately, the application that had been bought did not allow such access. An alternative scenario might have been that Brigette identified this as a possible risk early on in the project. She might have come across Richard Fairley's four COTS (commercial off-the-shelf) software acquisition risks – see Table 7.5 – where one risk is difficulty in integrating the data formats and communication protocols of different applications. Brigette might have specified that the selected package must use a widely accepted data management system like Oracle that allows easier integration.

TABLE 7.5 Fairley's four commercial off-the-shelf (COTS) software acquisition risks

1 Integration	Difficulties in integrating the data formats and communication protocols of different applications.
2 Upgrading	When the supplier upgrades the package, the package might no longer meet the users' precise requirements. Sticking with the old version could mean losing the supplier's support for the package.
3 No source code	If you want to enhance the system, you might not be able to do so as you do not have access to the source code.
4 Supplier failures or buyouts	The supplier of the application might go out of business or be bought out by a rival supplier.

See R. Fairley (1994) 'Risk management for software projects' *IEEE Software* 11(3) 57–67.

Risk mitigation can sometimes be distinguished from risk reduction. *Risk reduction* attempts to reduce the likelihood of the risk occurring. *Risk mitigation* is action taken to ensure that the impact of the risk is lessened when it occurs. For example, taking regular back-ups of data storage would reduce the impact of data corruption but not its likelihood. Mitigation is closely associated with contingency planning which is discussed presently.

Risk transfer

In this case the risk is transferred to another person or organization. With software projects, an example of this would be where a software development task is outsourced to an outside agency for a fixed fee. You might expect the supplier to quote a higher figure to cover the risk that the project takes longer than the 'average' expected time. On the other hand, a well-established external organization might have productivity advantages as its developers are experienced in the type of development to be carried out. The need to compete with other software development specialists would also tend to drive prices down.

Risk transfer is what effectively happens when you buy insurance.

7.8 Risk Management

Contingency

Risk reduction activities would appear to have only a small impact on reducing the probability of some

risks, for example staff absence through illness. While some employers encourage their employees to adopt a healthy lifestyle, it remains likely that some project team members will at some point be brought down by minor illnesses such as flu. These kinds of risk need a contingency plan. This is a planned action to be carried out if the particular risk materializes. If a team member involved in urgent work were ill then the project manager might draft in another member of staff to cover that work.

The preventative measures that were discussed under the 'Risk reduction' heading above will usually incur some cost regardless of the risk materializing or not. The cost of a contingency measure will only be incurred if the risk actually materializes. However, there may be some things that have to be done in order for a contingency action to be feasible. An obvious example is that back-ups of a database have to be taken for the contingency action when the database is corrupted is to restore it from back-ups. There would be a cost associated with taking the back-ups.

EXERCISE 7.4

In the case above where staff could be absent through illness, what preconditions could facilitate contingency actions such as getting other team members to cover on urgent tasks? What factors would you consider in deciding whether these preparatory measures would be worthwhile?

Deciding on the risk actions

Five generic responses to a risk have been discussed above. For each actual risk, however, specific actions have to be planned. In many cases experts have produced lists recommending practical steps to cope with the likelihood of particular risks; see, for example, Boehm's 'top ten' software engineering risks in Table 7.1.

Whatever the countermeasures that are considered, they must be cost-effective. On those occasions when a risk exposure value can be calculated as a financial value using the (*value of damage*) × (*probability of occurrence*) formula – recall Section 7.6 – the cost-effectiveness of a risk reduction action can be assessed by calculating the *risk reduction leverage* (*RRL*).

$$\textit{risk reduction leverage} = (RE_{before} - RE_{after})/(\textit{cost of risk reduction})$$

RE_{before} is the risk exposure, as explained in Section 7.6, before risk reduction actions have been taken. RE_{after} is the risk exposure after taking the risk reduction action. An RRL above 1.00 indicates that the reduction in risk exposure achieved by a measure is greater than its cost. To take a rather unrealistic example, it might cost £200,000 to replace a hardware configuration used to develop a software application. There is a chance of a fire (because of the particular location of the installation, say). The risk exposure would be of £200,000, that is £2,000. Installing fire alarms at a cost of £500 would reduce the chance of fire to 1%. The new risk exposure would be £1,000, a reduction of £1,000 on the previous exposure. The RRL would be (2000 – 1000)/500, that is 2.0, and the action would therefore be deemed worthwhile.

Earlier in this chapter, we likened risk exposure to the amount you might pay to an insurance company to cover a risk. To continue the analogy, an insurance company in the above example might be willing to reduce the premium you pay to have cover against fire from £2,000 to £1,000 if you installed fire alarms. As the fire alarms would cost you only £500 and save £1,000, the cost would clearly be worthwhile.

EXERCISE 7.5

Assume that the likelihood of one of your valuable team members leaving the project midway is 0.5. In case the member actually leaves, there is a 25% chance that the project would miss the delivery date. You consider the customer's consequent displeasure to be equivalent to £50,000 in monetary terms. To counter the risk, you can recruit a fresh engineer at a salary of £2000 per month for six months, to essentially act as a back-up for the valuable team member. Also, assume that the contribution of the back-up engineer to the project, if the regular engineer does not leave, would be 0.2 of the employment duration. After employing the back-up engineer, the probability of missing the project deadline is expected to be only about 10%. Would it be a good idea to employ the back-up engineer?

Creating and maintaining the risk register

When the project planners have picked out and examined what appear to be the most threatening risks to the project, they need to record their findings in a *risk register*. The typical content of such a register is shown in Figure 7.5. After work starts on the project more risks will emerge and be added to the register. At regular intervals, probably as part of the project control life cycle described in Chapter 9, the risk register should be reviewed and amended. Many risks threaten just one or two activities, and when the project staff have completed these the risk can then be 'closed' as no longer relevant. In any case, as noted earlier, the probability and impact of a risk are likely to change during the course of the project.

7.9 Evaluating Risks to the Schedule

In Section 7.6 we showed a probability chart – Figure 7.3. This illustrated the point that a forecast of the time needed to do a job is most realistically presented as a graph of likelihood of a range of figures, with the most likely duration as the peak and the chances of the job taking longer or shorter shown as curves sloping down on either side of the peak. Thus we can show that a job might take five days but that there is a small chance it might need four or six days, and a smaller chance of three or seven days, and so on. If a task in a project takes longer than planned, we might hope that some other task might take less and thus compensate for this delay. In the following sections we will examine PERT, a technique which takes account of the uncertainties in the durations of activities within a project. We will also touch upon Monte Carlo simulation, which is a more powerful and flexible tool that tackles the same problem.

A drawback to the application of methods like PERT is that in practice there is a tendency for developers to work to the schedule even if a task could be completed more quickly. Even if tasks are completed earlier than planned, project managers are not always quick to exploit the opportunities to start subsequent activities earlier than scheduled. Critical chain management will be explored as a way of tackling this problem.

7.10 Applying the PERT Technique

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.

RISK RECORD

Risk id		Risk title	
Owner		Date raised	Status

Risk description

Impact description

Recommended risk mitigation

Probability/impact values

	Probability	Impact		
		Cost	Duration	Quality
Pre-mitigation				
Post-mitigation				

Incident/action history

Date	Incident/action	Actor	Outcome/comment

FIGURE 7.5 Risk register page

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.

The method is very similar to the CPM technique (indeed many practitioners use the terms PERT and CPM interchangeably) but, instead of using a single estimate for the duration of each task, PERT requires three estimates.

- *Most likely time*: the time we would expect the task to take under normal circumstances. We shall identify 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* for 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 call this *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 7.6

Table 7.6 provides additional activity duration estimates for the network shown in Figure 6.29. 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.

TABLE 7.6 PERT activity time estimates

Activity	Optimistic (*a*)	Activity durations (weeks). 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

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 the dates by which we expect to achieve those events.

EXERCISE 7.7

Before reading further, use your calculated expected activity durations to carry out a forward pass through the network (Figure 6.29) 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 7.6 indicates that we expect the project to take 13.5 weeks. In Figure 7.6 we have used an activity-on-arrow network as this form of presentation makes it easier to separate visually the estimated activity data (expected durations and, later, their standard deviations) from the calculated data

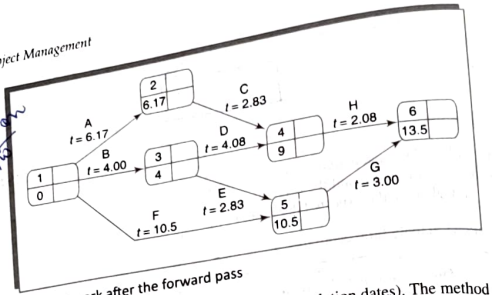


FIGURE 7.6 The PERT network after the forward pass

Even number	Target date
Expected date	Standard deviation

The PERT event labelling convention adopted here indicates event number and its target date along with the calculated values for expected time and standard deviation.

(expected completion dates and target completion dates). The method can, of cour be equally well supported by activity-on-node diagrams.

Unlike the CPM approach, the PERT method does not indicate the earliest a by which we could complete the project but the expected (or most likely) date. advantage of this approach is that it places an emphasis on the uncertainty of the world. Rather than being tempted to say '*the completion date for the project is . . .* are led to say '*we expect to complete the project by . . .*'

It also focuses attention on the uncertainty of the estimation of activity duration Requesting three estimates for each activity emphasizes the fact that we are certain what will happen – we are forced to take into account the fact that estim are approximate.

Activity standard deviations

A quantitative measure of the degree of uncertainty of an activity duration estimate may be obtained by ca lating the standard deviation s of an activity time, using the formula

$$s = \frac{b - a}{6}$$

This standard deviation formula is based on the rationale that there are approximately six standard deviations between the extreme tails of many statistical distributions.

The activity standard deviation is proportional to the difference between the optim and pessimistic estimates, and can be used as a ranking measure of the degr uncertainty or risk for each activity. The activity expected durations and stan deviations for our sample project are shown in Table 7.7.

The likelihood of meeting targets

The main advantage of the PERT technique is that it provides a method for estimating the probab meeting or missing target dates. There might be only a single target date – the project completion – b might wish to set additional intermediate targets.

TABLE 7.7 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

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, which must take place by week 10. These three target dates are shown on the PERT network in Figure 7.7.

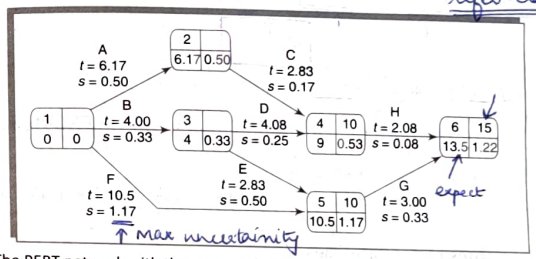


FIGURE 7.7 The PERT network with three target dates and calculated event standard deviations

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

The square of the standard deviation is known as the variance. Standard deviations may not be added together but variances may.

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 7.8 illustrates the technique. One practical outcome of this is that the contingency time to be allocated to a sequence of activities as a whole would be less than the sum of the contingency allowances for each of the component activities. This has implications that can be exploited in critical chain project management, which are discussed in the next section.

EXERCISE 7.8

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 7.7.

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 7.9

The *z* value for event 4 is (10 – 9.00)/0.53 = 1.8867.

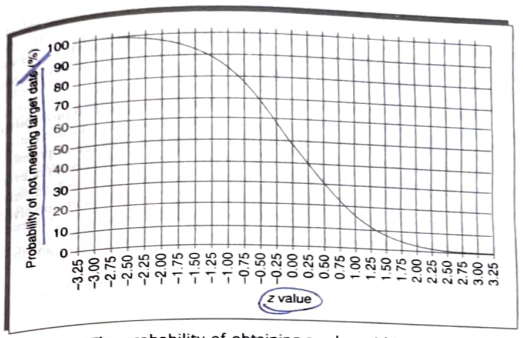
Calculate the *z* values for the other events with target dates in the network shown in Figure 7.7.

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 7.8.

EXERCISE 7.10

The *z* value for the project completion (event 6) is 1.23. Using Figure 7.8 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.



This graph is the equivalent of tables of *z* values, also known as standard normal deviates, which may be found in most statistics textbooks.

FIGURE 7.8 The probability of obtaining a value within *z* standard deviations of the mean for a normal distribution

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 multi-valued 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 regarding 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.

7.11 Monte Carlo Simulation

As an alternative to the PERT technique, we can use Monte Carlo simulation approach. Monte Carlo simulation are a class of general analysis techniques that are valuable to solve any problem that is complex, nonlinear, or involves more than just a couple of uncertain parameters. Monte Carlo simulations involve repeated random sampling to compute the results. Since this technique is based on repeated computation of random numbers, it becomes easier to use this technique when available as a computer program.

When Monte Carlo simulation is used to analyse the risk of not meeting the project deadline, the project completion time is first modelled as a mathematical expression involving the probability distributions of the completion times of various project activities and their precedence relationships. Activity durations 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 as shown in the probability chart in Figure 7.4, we might be able to specify durations as pertinent probability distributions. With less information available, we should, at least, be able to provide three time estimates as used by PERT.

Monte Carlo simulation essentially evaluates a range of input values generated from the specified probability distributions of the activity durations. It then calculates the results repeatedly; each time using a different set of random values generated from the given probability functions. Depending upon the number of probabilistic parameters and the ranges specified for them, a Monte Carlo simulation could involve thousands or even millions of calculations to complete. After the simulation results are available, these are analysed, summarized and represented graphically, possibly using a histogram as shown in Figure 7.9. The main steps involved in carrying out Monte Carlo simulation for a project consisting of n activities are as follows:

- *Step 1:* Express the project completion time in terms of the duration of the n activities ($x_{i,\ i=1,\ n}$ and their dependences as a precedence graph, $d = f(x_1, x_2, \ldots, x_n)$.
- *Step 2:* Generate a set of random inputs, $x_{i1}, x_{i2}, \ldots, x_{in}$ using specified probability distributions.
- Step 3: Evaluate the project completion time expression and store the result in d_i.
- *Step 4:* Repeat Steps 2 and 3 for the specified number of times.
- *Step 5:* Analyze the results $d_{i,\ i=1,n}$: summarize and display using a histogram as the one shown in Figure 7.9.

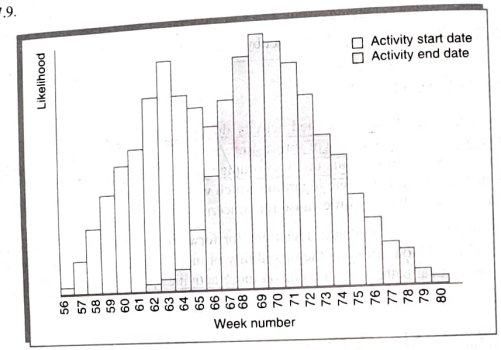


FIGURE 7.9 Risk profile for an activity generated using Monte Carlo simulation

To appreciate the advantage of Monte Carlo simulations over a manual approach, consider the following. In the manual approach, a few combinations of each project duration are chosen (such as best case, worst case, and most likely case), and the results recorded for each selected scenario. In contrast, in Monte Carlo simulation, hundreds or thousands of possible random sampling of probability distribution functions of the

activity durations are considered as samples for evaluation of the project completion time expression to produce outcomes. Monte Carlo simulation is expected to give a more realistic result than manual analysis of a few cases, especially because manual analysis implicitly gives equal weights to all scenarios.

7.12 Critical Chain Concepts

This chapter has stressed the idea that the forecast for the duration of an activity cannot in reality be a single number, but must be a range of durations that can be displayed on a graph such as Figure 7.3. However, we would want to pick one value in that range which would be the *target*.

The duration chosen as the target might be the one that seems to be the *most likely*. Imagine someone who cycles to work each day. It may be that on average it takes them about 45 minutes to complete the journey, but on some days it could be more and on others it could be less. These journey times could be plotted on a graph like the one in Figure 7.3. If the cyclist had a very important meeting at work, it is likely that they would give themselves more time – say an extra 15 minutes – than the average 45 minutes to make sure that they arrived in time. In the discussion above on the PERT risk technique the most likely duration was the middle value and the pessimistic estimate was the equivalent of the 45 + 15 = 60 minutes.

Of course, there will be some days when the cyclist will beat the average of 45 minutes. When a project is actually being executed, the project manager will be forced to focus on the activities where the actual durations exceed the target. Activities which are actually completed before the target date are likely to be overlooked. These early completions, properly handled, could put some time in hand that might still allow the project to meet its target completion date if the later activities are delayed.

Figure 7.10 shows the findings of Michiel van Genuchten, a researcher who analysed the reasons for delays in the completion of software development tasks. This bar chart shows that about 30% of activities were finished on time, while 9% were a week early and 17% were a week late. The big jump of 21 percentage points between being a week early and being on time is compatible with the 'Parkinson's Law' principle that '*work expands to fill the time available*'. This tendency should not be blamed on inherent laziness. van Genuchten found that the most common reason for delay was the time that had to be spent on non-project work. It seems that developers used spare time provided by generous estimates to get on with other urgent work.

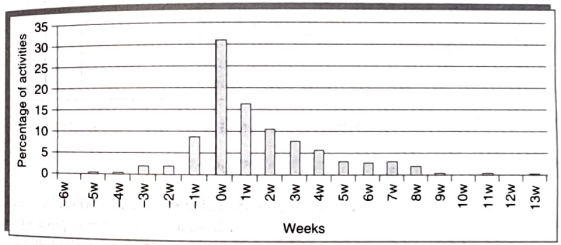


FIGURE 7.10 Percentage of activities early or late (after van Genuchten, 1991)

Michiel van Genuchten (1991) 'Why is software late? An empirical study of reasons for delay in software development', *IEEE Transactions in Software Engineering* 17(6) 582–90.

A good introduction is L. P. Leach (1999) 'Critical chain project management improves project performance' *Project Management Journal* 30(2) 39–51.

One approach which attempts to solve some of these problems is the application of the *critical chain* concept originally developed by Eliyahu Goldratt. In order to demonstrate the principles of this approach, the example shown in Figure 7.7 will be reworked as a Gantt chart. Figure 7.11 shows what the Gantt chart for this project might look like if a 'traditional approach' were adopted, but we have already adopted the most likely durations.

The general steps in the Critical Chain approach are explained in the following sections.

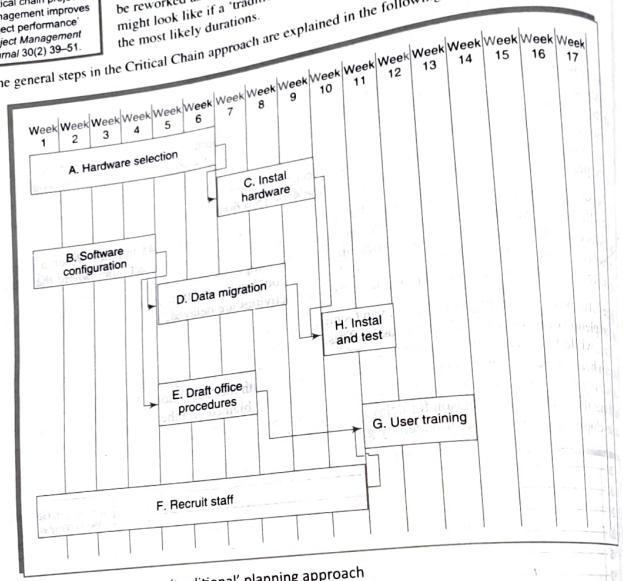


FIGURE 7.11 Gantt chart – 'traditional' planning approach

Deriving 'most likely' activity durations

The target date generated by critical chain planning is one where it is estimated that there is a 50% chance of success – this approximates to the expected time identified in the PERT risk method. In some explanations of critical chain project planning it is suggested that the most likely activity duration can be identified by halving the estimates provided. This is based on the assumption that the estimates given to the planner will be 'safe' ones based on a 95% probability of them being achieved. If you look at Figure 7.3, the 95% estimate would be 9 days and half of that (4.5 days) would not be a reasonable target as it would have a probability

only 10% of success. It also assumes that a probability profile has a bell-shaped normal distribution (like the example in Figure 7.3). If you look at the distribution which resulted from van Genuchten's research – see Figure 7.10 – you can see that it is certainly not bell-shaped. Other critical chain experts suggest deducting 33% from the safe estimate to get the target estimate – which seems less unreasonable.

However, what appear to be arbitrary managerial reductions in the estimates may not be a good way to motivate developers, especially if these staff supplied the estimates in the first place. A better approach would be to ask developers to supply two estimates. One of these would be a 'most likely' estimate and the other would include a safety margin or comfort zone. From now on we are going to assume that this is what has happened. In fact we will use the figures already presented in Table 7.6 in this new role (Table 7.8).

TABLE 7.8 Most likely and comfort zone estimates (days)

Activity	Most likely	Plus comfort zone	Comfort zone
A	6	8	2
B	4	5	1
C	3	3	0
D	4	5	1
E	3	4	1
F	10	15	5
G	3	4	1
H	2	2.5	0.5

Using latest start dates for activities

Working backwards from the target completion date, each activity is scheduled to start as late as possible. Among other things, this should reduce the chance of staff being pulled off the project on to other work. It is also argued – with some justification according to van Genuchten's research above – that most developers would tend to start work on the task at the latest start time anyway. However, it does make every activity 'critical'. If one is late the whole project is late. That is why the next steps are needed.

Inserting project and feeder buffers

To cope with activity overruns, a *project buffer* is inserted at the end of the project before the target completion date. One way of calculating this buffer is as the equivalent of 50% of the sum of lengths of the 'comfort zones' that have been removed from the *critical chain*. The critical chain is the longest chain of activities in the project, taking account of both task and resource dependencies. This is different from the *critical path* as the latter only takes account of task dependencies. A *resource dependency* is where one activity has to wait for a resource (usually a person in software development) which is being used by another activity to become available. If an activity on this critical chain is late it will push the project completion date further into the project buffer. That the buffer should be 50% of the total comfort zones for critical chain activities is based

on the grounds that if the estimate for an activity was calculated as having a 50% chance of being correct, the buffer would only need to be called upon by the 50% of cases where the estimate was not correct.

An alternative proposal is to sum the squares of the comfort zones and then take the square root of the total. This is based on the idea that each comfort zone is the equivalent to the standard deviation of the activity. If you back and look at the section headed *Calculating the standard deviation of each project event* in Section 7, this method of calculation still produces a figure which is less than simply summing all the comfort zones. This is justified on the grounds that the contingency time needed for a group of activities is less than the sum of the individual contingency allowances as the success of some activities will compensate for the shortfalls in others.

Buffers are also inserted into the project schedule where a subsidiary chain of activities feeds into the critical chain. These *feeding buffers* could once again be set at 50% of the length of the 'comfort zone' removed from the subsidiary or *feeding chain*.

A worked example

Figure 7.12 shows the results of this process. The critical chain in this example happens to be the same as the critical path, that is activities F and G which have comfort zones of 5 weeks and 1 week respectively, making a total of 6 weeks. The project buffer is therefore 3 weeks.

Subsidiary chains feed into the critical chain where activity H links into the project buffer and where activity E links into G which is part of the critical chain. Feeding buffers are inserted at these points. For the first buffer the duration would be 50% of the saved comfort zones of A, C and H, that is $(2 + 0 + 0.5)/2 = 1.25$ weeks. It could be argued that B, D and H could form a feeder chain which also has a combined comfort zone of $(1 + 1 + 0.5)/2 = 1.25$ weeks. In the situations where there are parallel alternative paths on a feeder chain, the practice is to base the feeding buffer on whichever comfort zone total is greater. This because if one or other or both parallel paths were late they could still use the same buffer. (Imagine that in the example above there are two cyclists who live 45 minutes away from work and they both have the same important meeting – they might each add a 15-minute comfort zone to the ride on that day but that 15 minutes could effectively be the same 15 minutes between 7.45 and 8.00 a.m. in the morning). It could be argued that the feeding buffer and the final project buffer could also be merged, but explanations of critical chain planning such as that of Larry Leach (see above), make clear that this is not to be done. This could be because a delay penetrating the feeding buffer time does not affect the completion date of the project, while penetrating the project buffer does.

In the second place, where a feeder chain of activities joins the critical chain, the feeder buffer would be 50% of the comfort zones of activities B and E, that is 1 week.

Project execution

When the project is executed, the following principles are followed:

- No chain of tasks should be started earlier than scheduled, but once it has been started it should be finished as soon as possible – this invokes the *relay race principle*, where developers should be ready to start their tasks as soon as the previous, dependent, tasks are completed.
- Buffers are divided into three zones: green, amber and red, each of an even (33%) size:
  - *green*, where no action is required if the project completion date creeps into this zone;

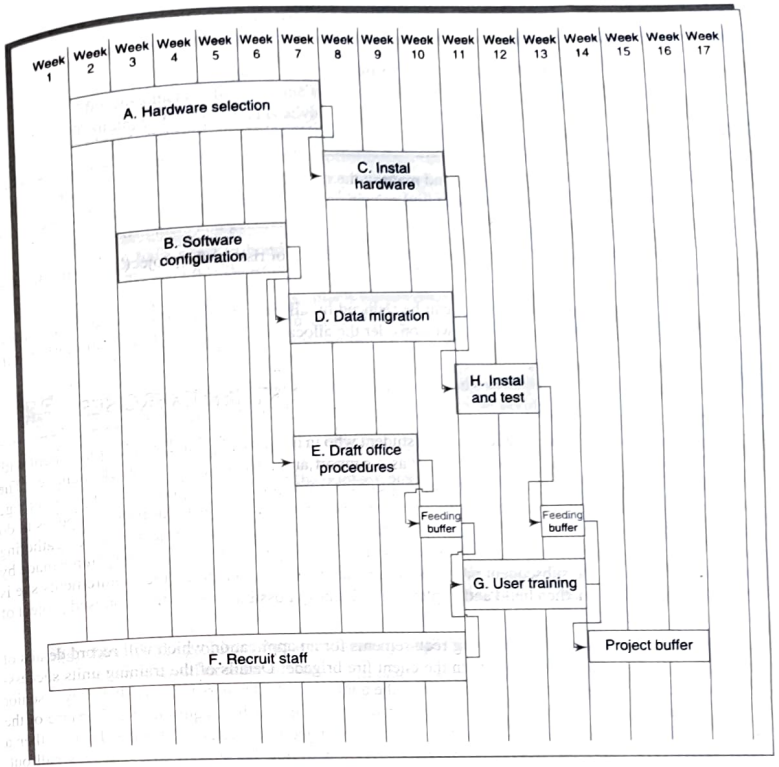


FIGURE 7.12 Gantt chart – critical chain planning approach

  - *amber*, where an action plan is formulated if the project completion dates moves into this zone;
  - *red*, where the action plan above is executed if the project penetrates this zone.

Critical chain planning concepts have the support of a dedicated group of enthusiasts. However, the full application of the model has attracted controversy on various grounds. Our personal view is that the ideas of:

- requiring two estimates: the most likely duration/effort and the safety estimate which includes additional time to deal with problems that could arise with the task, and

See, for example, D. Trietsch (2005) 'Why a critical path by any other name would smell less sweet' *Project Management Journal* 36(1) 27–36 and T. Raz et al. (2003) 'A critical look at critical chain project management' *Project Management Journal* 34(4) 24–32.

- placing the contingency time, based on the 'comfort zone' which is the difference between the
likely and safety estimates, in common buffers rather than associating it with individual activities

are sound ones that could usefully be absorbed into software project management practice.

CONCLUSION

In this chapter, we have seen how to identify and manage the risks that might affect the success of a pro
Risk management is concerned with assessing and prioritizing risks and drawing up plans for addres
those risks before they become problems.

This chapter has also described techniques for estimating the effect of risk on the project's activity net
and schedule.

Many of the risks affecting software projects can be reduced by allocating more experienced staff to
activities that are affected. In the next chapter we consider the allocation of staff to activities in more de

FURTHER EXERCISES

1. Fiona is a final-year computing undergraduate student who in her third year undertook a placement
the ICT department of an insurance company as a support analyst and then a network manager
placement year was very busy and rewarding as the company saw ICT as providing business adv
in what was a very dynamic and aggressively competitive sector. The project that Fiona proposes
in her final year will use the insurance company as a client. The proposed project involves gath
requirements for an application that records details of change requests for operational systems mai
users and then tracks the subsequent progress of the change. Having gathered the requirements
to design the application, then build and implement it. Identify possible risks in the proposed proj
which Fiona should take account.
2. Mo is a systems analyst who is gathering requirements for an application which will record deta
the training undertaken by fire-fighters in the client fire brigade. Details of the training units
fully completed by fire-fighters are to be input to the application by trainers who are themselves
and active fire-fighters. Mo needs to interview a trainer to obtain his/her requirements. Because
senior fire-fighters' other duties the interview has to be arranged two weeks in advance. There is
20% chance of the fire-fighter being unable to attend the interview because of an emergency ca
Each week that the project is delayed costs the fire brigade approximately £1,000.
   (a) Provide an estimate of the risk exposure (as a financial value) for the risk that the senio
   fighter might not be able to attend at the times needed.
   (b) Suggest possible risk mitigation actions.
3. In Exercise 7.2 you were asked to identify risks under the four headings of Actors, Technology, Str
and Tasks for the IOE annual maintenance contracts and the Brightmouth College payroll scen
Now identify risks for each scenario that relate to pairs of domains, for example Actors–Techn
Actors–Tasks, and so on.
4. The Wet Holiday Company specializes in the provision of holidays which involve water sp
various types. There are three major divisions with the following lines of business:
   - boat holidays on canals;
   - villa holidays in various parts of the Mediterranean which involve sailing in some way;
   - canoeing holidays in France.

Wet Holiday feel that they are particularly appealing to a young active market and that having the facility for customers to book via the web is essential. They call in ICT consultants to advise them on their IT strategy. The consultants advise them that before they can have a web presence, they need to have a conventional ICT-based booking system to support their telesales operation first. Because of the specialized nature of their business, an off-the-shelf application would not be suitable and they would need to have a specially written software application, based on a client–server architecture. The top priority needs to be given to a system to support villa holiday bookings because this has the largest number of customers and generates the most revenue.

Wet Holiday have some in-house IT development staff, but these are inexperienced in client–server technology. To meet this shortfall, contractors are employed.

It turns out that development takes much longer than planned. Much of this delay occurs at acceptance testing when the users find many errors and performance shortcomings, which require extensive rework. Part of the problem relates to getting the best performance out of the new architecture; this has a particular impact on response times which are initially unacceptable to staff who are dealing with customers over the phone. The contractors are not closely monitored and some of the code that they produce is found to have many careless mistakes and to be poorly structured and documented. This makes it difficult to make changes to the software after the contractors have left on the expiry of their contracts.

The villa booking system can only be implemented at the beginning of a holiday season and the deadline for the beginning of the 2002 to 2003 season is missed, leading to a 12-month delay in the implementation. The delay in implementation seems to encourage the users to ask for further modifications to the original requirements, which adds even more to development costs.

The delays in implementing this application mean that the other scheduled IT development, for other lines of business, have to be put back. Managers of customer-facing business functions at Wet Holiday are suggesting that the whole IT function should be completely outsourced.

   (a) Identify the problems that were faced by Wet Holiday, and describe actions that could have been taken to avoid or reduce them.
   (b) Use your findings in (a) to create a risk checklist for future projects.
5. Below are details of a project. All times are in days.

Activity	Depends on	Optimistic time	Most likely	Pessimistic
A	–	8	10	12
B	A	10	15	20
C	B	5	7	9
D	–	8	10	12
E	D,C	3	6	9

Using the activity times above:

- Calculate the expected duration and standard deviation for each activity.
- Identify the critical path.
- Draw up an activity diagram applying critical chain principles for this project:
  - Locate the places where buffers will need to be located.
  - Assess the size of the buffers.
  - Start all activities as late as possible.

6. Below are details of a project. All times are in days.

Activity	Depends on	Most likely	Plus safety
A		10	14
B	A	5	7
C	B	15	21
D	A	3	5
E	A	8	12
F	E	20	22
G	D	6	8
H	C,F,G	10	14

   (a) Using (i) the most likely and then (ii) the safety estimates:
   - Calculate the earliest and latest start and end days and float for each activity.
   - Identify the critical path.

   (b) Draw up an activity diagram applying critical chain principles for this project:
   - Locate the places where buffers will need to be located.
   - Assess the size of the buffers.
   - Start all activities as late as possible.

7. In this chapter the application of risk management to software development projects has been st advocated. In practice, however, managers are often reluctant to apply the techniques. What t think might be the reasons for this?

8. Suppose you are the project manager of a large software development project. List three commo of risks that your project might suffer. Point out the main steps that you would follow to effec manage risks in your project.

9. Schedule slippage is a very common form of risk that almost every project manager has to dea Suppose you are the project manager of a medium-sized project. Explain how you would mana risk of schedule slippage.

8

RESOURCE ALLOCATION

OBJECTIVES

When you have completed 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:

- an activity schedule indicating the planned start and completion dates for each activity;

These schedules will provide the basis for the day-to-day control and management of the project. These are described in Chapter 9.