Unit 3

4.SELECTION OF AN APPROPRIATE PROJECT APPROACH

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.

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.

Build or Buy?

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.

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.

An option which is increasingly taken is to obtain a licence to run off-the-shelf software.

Advantages

- 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
    1. it can be examined and perhaps even trialed before acquisition,
    2. 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.

Disadvantages

- 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 fi t 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.

Choosing Methodologies and Technologies

The term methodology describes a collection of methods. 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.

Project analysis should select the most appropriate methodologies and technologies for a project. Methodologies include approaches like the Unifi ed 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

The steps of project analysis.

- **Identify project as either objective-driven or product-driven**
  A product-driven project creates products defi ned before the start of the project. An objective-driven project will often have come first which will have defi ned the general software solution that is to be implemented.
- **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.
  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?
  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.
- **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.
  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.
  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.
  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.
- **Take into account user requirements concerning implementation**
  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 m
- **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.

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.

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.

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.

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.

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 computerbased 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.

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.

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 refi ned. Thus, over successive iterations, the full set of functionalities of the software takes shape.

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

Agile Method

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 any 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)

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.

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.
- 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.

Extreme Programming (XP)

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 the 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 specifi c 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.

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.

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, refl ect 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'. 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.

Refactoring A threat to the target of striving to have always the simplest design is that over time, as modifi cations 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.

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.

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 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.

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

- 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, where 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.

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 team 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.

Chapter 6- ACTIVITY PLANNING

A detailed plan for the project, however, must also include a schedule indicating the start and completion times for each activity. This will enable us to:

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

To be effective, a plan must be stated as a set of targets, the achievement or non-achievement of which can be unambiguously measured. The activity plan does this by providing a target start and completion date for each activity (or a window within which each activity may be carried out).

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.

The Objectives of Activity Planning

- **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.
- **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

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.

Project Schedules

Before work commences on a project or, possibly, a stage of a larger project, the project plan must be developed to the level of showing dates when each activity should start and fi nish and when and how much of each resource will be required. Once the plan has been refi ned to this level of detail we call it a project schedule.

Creating a project schedule comprises four main stages.

1. 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.
2. 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.
3. 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.
4. 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.

Projects and Activities

Defining activities

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

- A project is composed of a number of 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.

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.

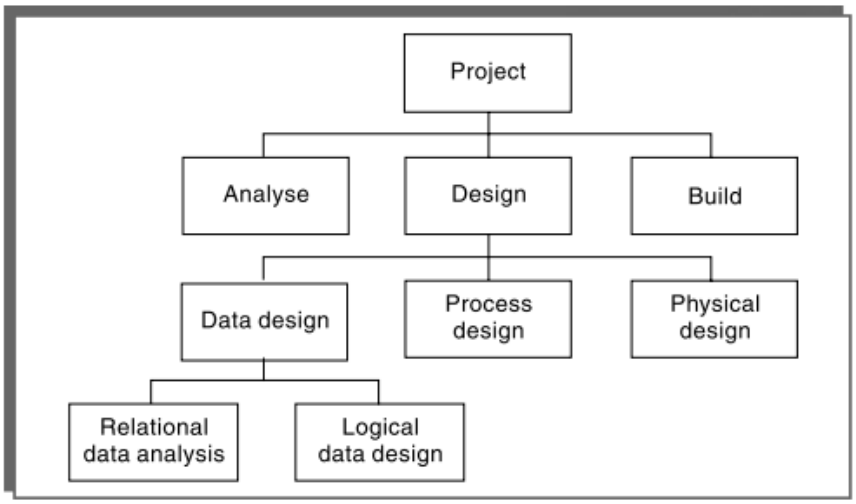


Fig – 6.2

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.

The WBS also represents a structure that may be refi ned 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.

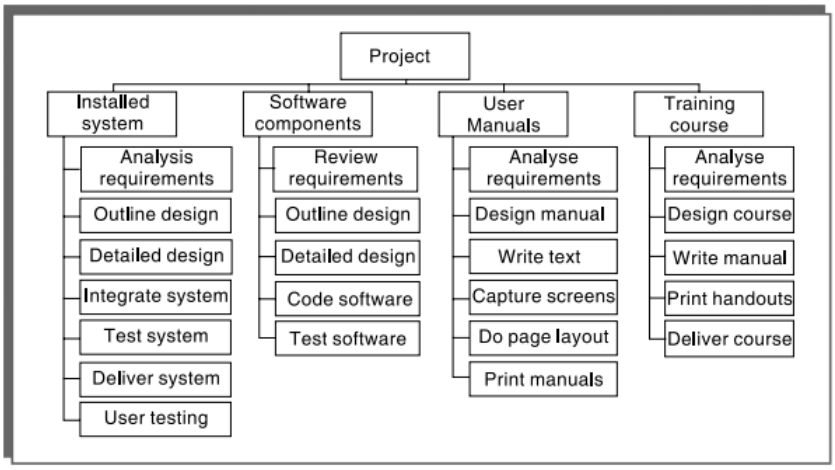
The product-based approach

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.

This approach is particularly appropriate if using a methodology such as SSADM or USDP (Unifi ed Software Development Process), which clearly specifies, for each step or task, each of the products required and the activities required to produce it.

The hybrid approach

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



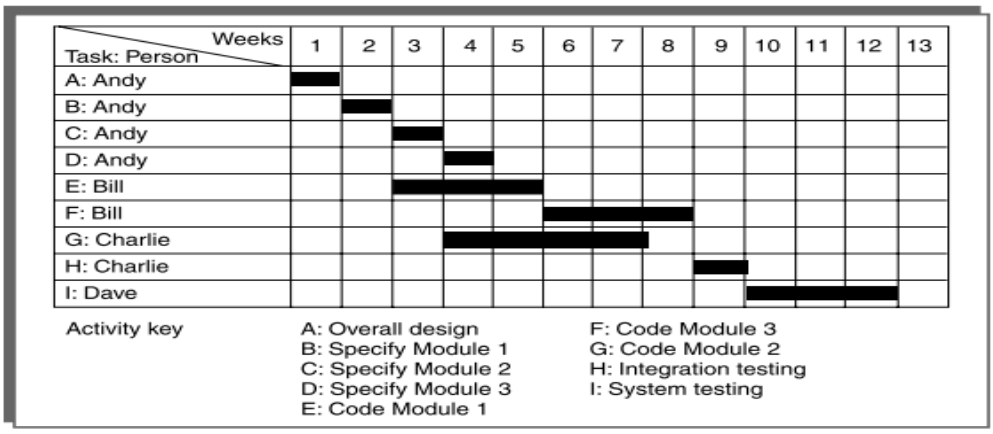
6.5 A hybrid Work Breakdown Structure based on deliverables and activities

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
- 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.

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.



6 A project plan as a bar chart

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.

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.

Network Planning Model

The 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.

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.

Rules for Network 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.
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.
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
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.

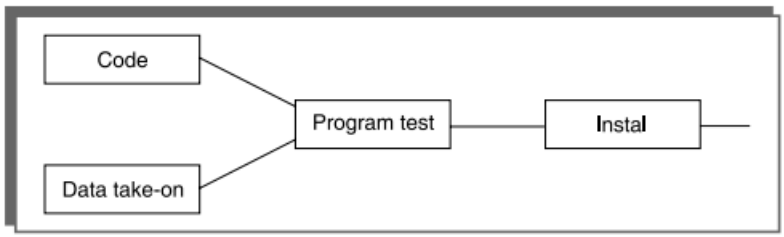


FIGURE 6.9 Fragment of a precedence network

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.

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.
A network may not contain loops 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.
A network should not contain dangles In many cases dangling activities indicate errors in logic when activities are added as an afterthought.

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 if evaluating a prototype. In such a case we could designate an activity 'test and document amendments'. This would, however, make it impossible to show that amendment recording could start, say, one day after testing had begun and fi nish a little after the completion of testing.

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 activity label is usually a code developed to uniquely identify the activity and may incorporate a project code. 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.

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.

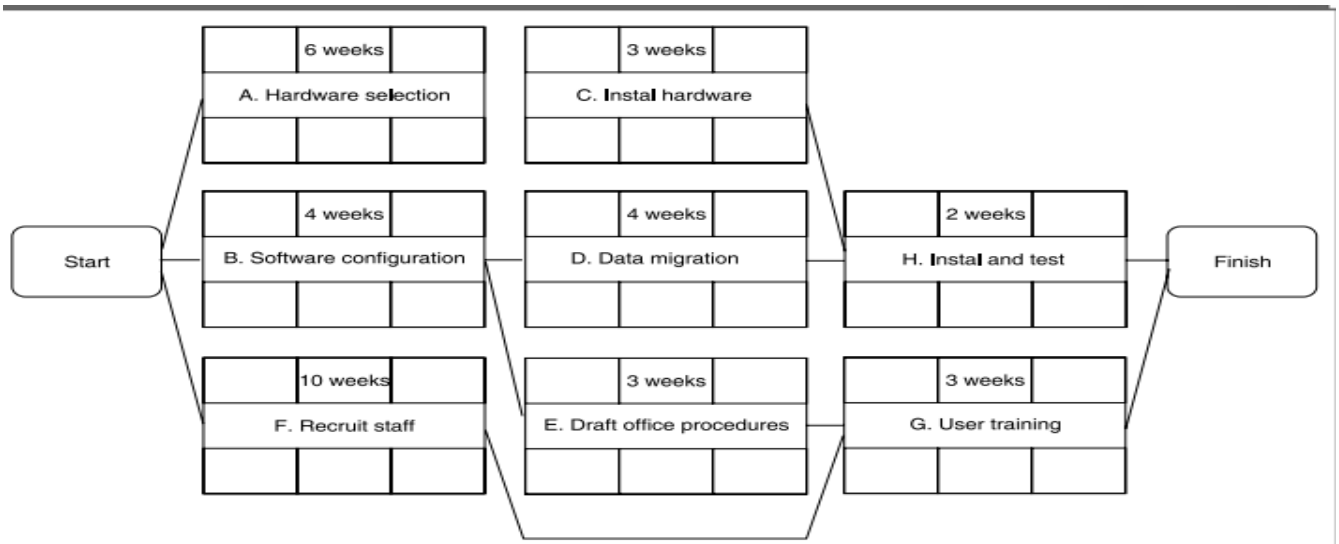
The method requires that for each activity we have an estimate of its duration. The network is then analysed by carrying out a forward pass, to calculate the earliest dates at which activities may commence and the project be completed, and a backward pass, to calculate the latest start dates for activities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

Th e Forward Pass

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

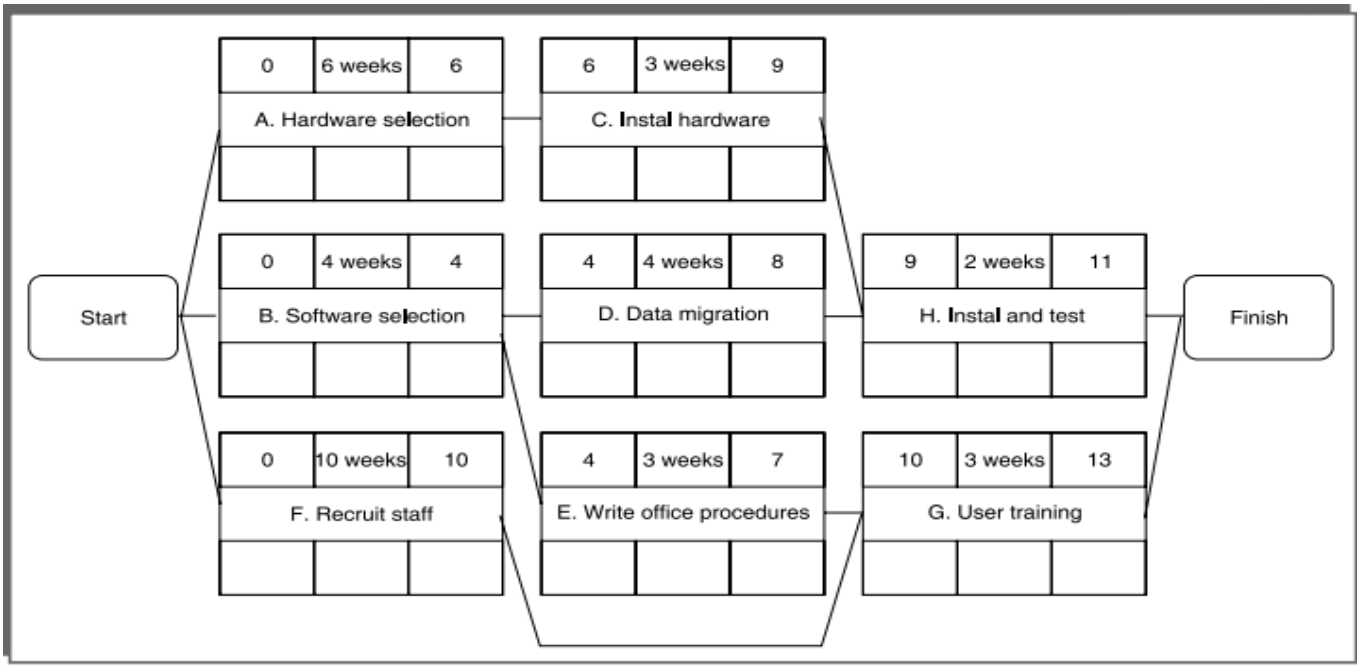
	Activity	Duration (weeks)	Precedents
A	Hardware selection	6	
B	System configuration	4	
C	Instal hardware	3	A
D	Data migration	4	B
E	Draft office procedures	3	B
F	Recruit staff	10	
G	User training	3	E, F
H	Instal and test system	2	C, D

The precedence network for the example project



- 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 fi nish is week 6.
- Activity B will take 4 weeks, so the earliest it can fi nish is week 4.
- Activity F will take 10 weeks, so the earliest it can fi nish 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 fi nish by week 8 and activity E, which will take 3 weeks, can therefore fi nish 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 fi nish 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.

The results of the forward pass



The Backward Pass

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 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 fi nish on time.

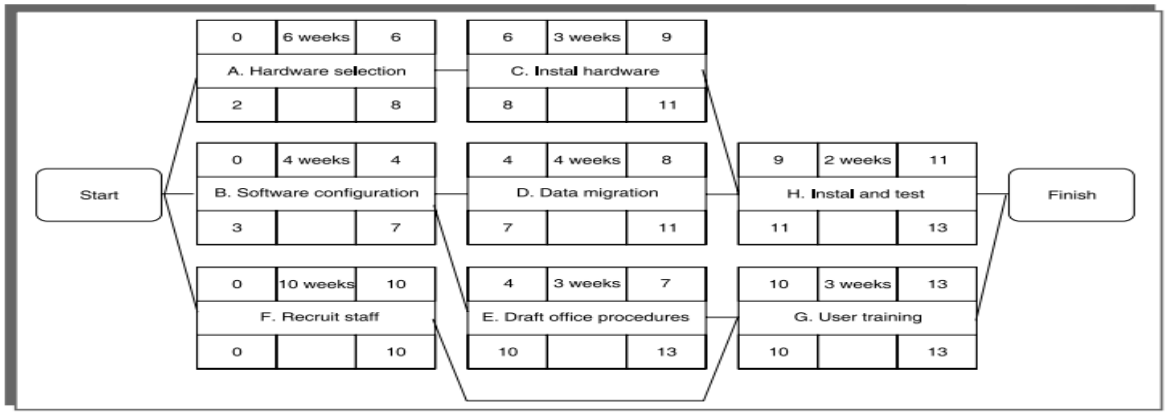


FIGURE 6.16 The network after the backward pass

Identifying the Critical Path

l 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 difference between an activity's earliest start date and its latest start date 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.

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 fi nish date and is a measure of the maximum time allowable for the activity

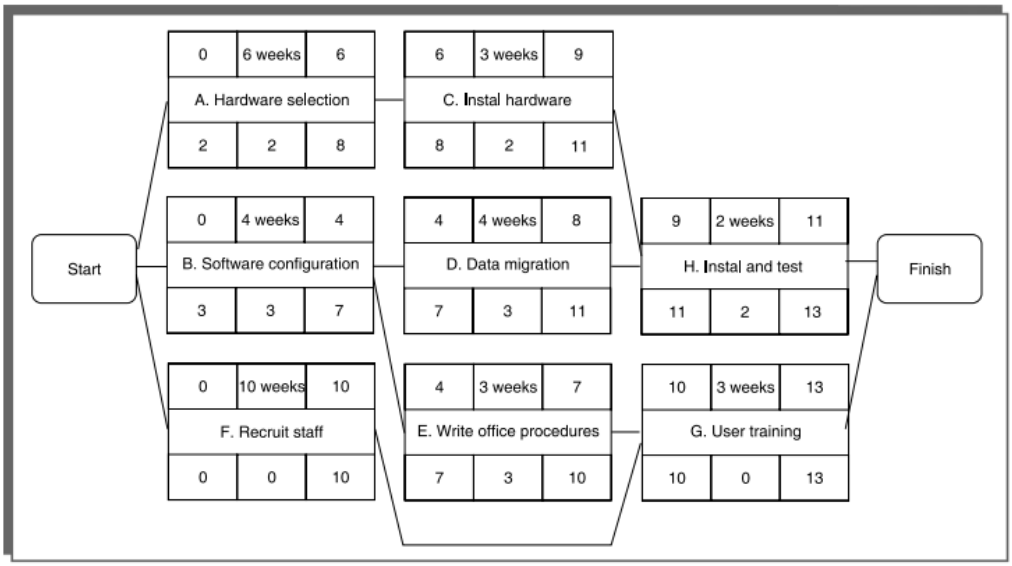


FIGURE 6.17 The critical path

Activity Float

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. If, however, activity A uses up its float (that is, it is not completed until week 8) then activity B will have zero float (it will have become critical). In such circumstances it may be misleading and detrimental to the project's success to publicize total float!

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

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

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.

Identifying Critical Activities

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

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

The importance of identifying critical and near-critical activities is that it is they that are most likely to be the cause of delays in completing the project.

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.

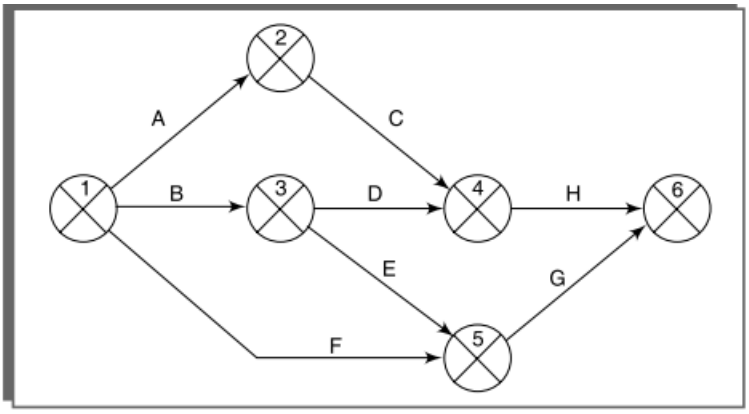


FIGURE 6.18 An activity-on-arrow network

Activity-on-arrow network rules and conventions

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.

A project network may have only one end node Again, this is a requirement for activity-on-arrow networks.

A link has duration A link represents an activity and, in general, activities take time to execute. Notice, however, that the network in Figure 6.18 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.

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.

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.

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

A network may not contain loops: loops are either an error of logic or a situation that must be resolved by itemizing iterations of activity groups.

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 fi rst 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.

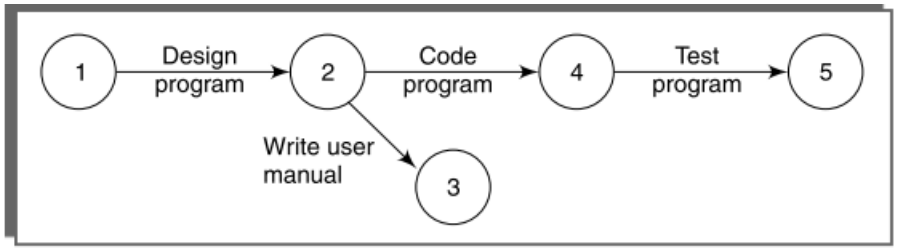


FIGURE 6.21 A dangle

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.

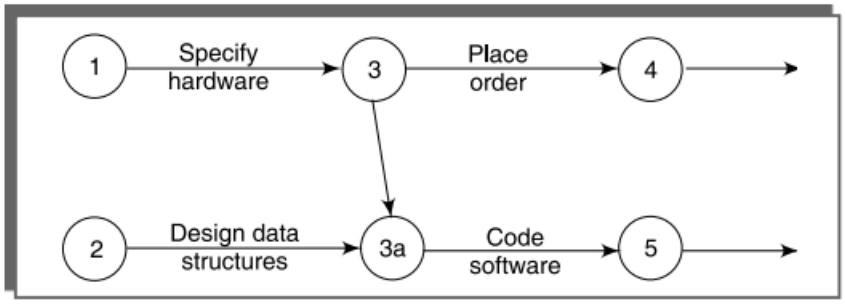


FIGURE 6.24 Two paths linked by a dummy activity

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

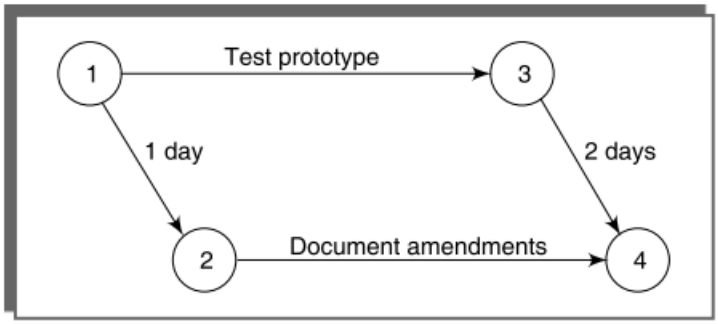
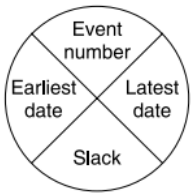


FIGURE 6.26 Using the ladder technique to indicate lags

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.

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).



Network analysis

Analysis proceeds in the same way as with activity-on-node networks, although the discussion places emphasis on the events rather than activity start and completion times.

The forward pass The forward pass is carried out to calculate the earliest date on which each event may be achieved and the earliest dates on which each activity may be started and completed. The earliest date for an event is the earliest date by which all activities upon which it depends can be completed. Using Figure 6.18 and Table 6.1, the calculation proceeds according to the following reasoning

- Activities A, B and F may start immediately, so the earliest date for event 1 is zero and the earliest start date for these three activities is also zero.
- Activity A will take 6 weeks, so the earliest it can fi nish is week 6 (recorded in the activity table). Therefore the earliest we can achieve event 2 is week 6.
- Activity B will take 4 weeks, so the earliest it can fi nish and the earliest we can achieve event 3 is week 4.
- Activity F will take 10 weeks, so the earliest it can fi nish 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 fi nish.
- 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 fi nish for activity D (week 8) and the earliest fi nish 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 fi nish for H) and 13 (the earliest fi nish for G).

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

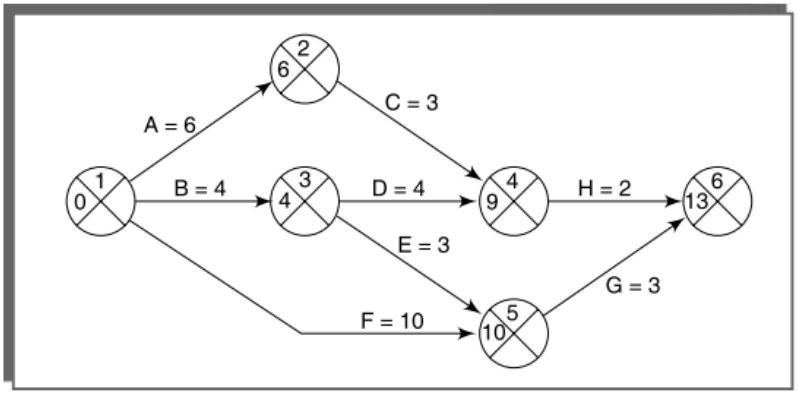


FIGURE 6.27 A CPM network after the forward pass

TABLE 6.3 The activity table after the forward pass

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

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

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

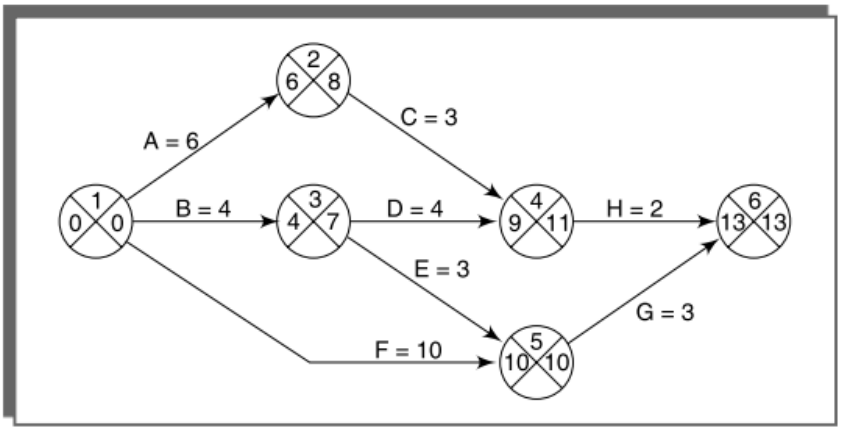


FIGURE 6.28 The CPM network after the backward pass

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 float
A	6	0	2	6	8	
B	4	0	3	4	7	
C	3	6	8	9	11	
D	4	4	7	8	11	
E	3	4	7	7	10	
F	10	0	0	10	10	
G	3	10	10	13	13	
H	2	9	11	11	13	

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).

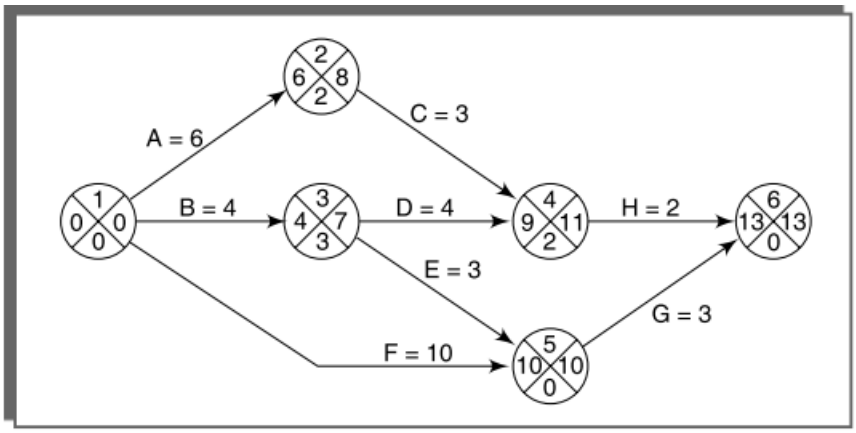


FIGURE 6.29 The critical path