# Chapter 10

# Information Systems Development and Project Management

## Learning Objectives

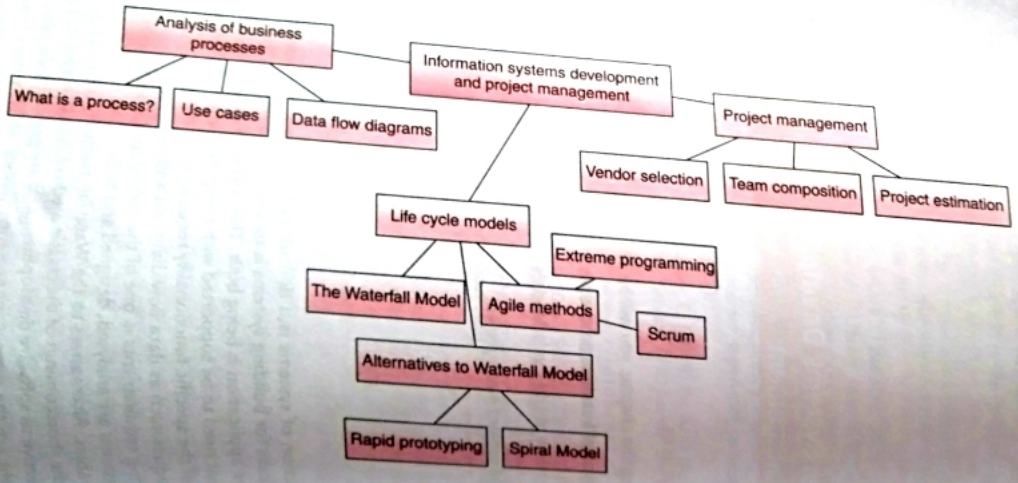
*After completing this chapter, you will be able to:*

- Learn analysis of business processes
- Get an overview of Life Cycle Models
- Understand project management

The construction of systems is a difficult task that requires careful analysis, design and planning. One of the first tasks for building systems is to understand the requirements or functions of the organisation for which the application is being built. This is known as analysis and entails understanding the business processes that need to be supported with computerisation, followed by a detailed exploration and documentation of the specific functionality required from the system. The analysis tasks can be performed by using techniques such as data flow diagrams and use cases.

Gathering requirements is one of the early steps in the Waterfall Model, which is the classic method by which systems are analysed, designed and built. It has well-defined steps, each of which has clear goals and deliverables. Alternatives to the Waterfall method, such as Rapid Prototyping and the Spiral Model, overcome the shortcomings of the classic method. Modern methods of agile programming bring an entirely new perspective to software development and are gaining wide popularity. Tied to the methods of systems development are the project management issues which ensure that the methods are indeed functional. Project management involves careful vendor selection, team composition and project estimation.

## MIND MAP OF CHAPTER TOPICS



## CASE STUDY: AMIS for Farmers in Bangladesh

One of the challenges farmers in Bangladesh face is that of getting the best possible price for their produce. Agricultural produce consists of food crops such as paddy and wheat, along with vegetables and fruit, and cash crops such as jute. When the produce is harvested, farmers have to find a wholesale market where they can sell it, as well as ensure that they get a good price for it to cover their investment and turn a profit.

The challenges that farmers face are formidable: Most live in remote areas without much access to transportation to markets, most are marginal farmers who eke out a living on small plots of land and cannot invest in resources to find the best markets for the produce, and many are illiterate and unable to access printed information. Furthermore, the traditional hold of moneylenders and local wholesalers is very strong preventing farmers from straying from local markets to seek better opportunities.

The problem of small and marginal farmers is common to the nations of the subcontinent, including India, Bangladesh, Nepal and Pakistan. Many small Indian farmers are suffering from a severe economic crisis: They are not able to recover the input costs for their crops from the revenues they obtain. This has led to large-scale farmer suicides in some regions, and in other regions farmers have simply given up farming and have moved to cities to work as labour.

When a young Bangladeshi researcher, Sirajul Islam, decided try and use information and communication technology (ICT) to alleviate the problems of farmers in Bangladesh, he faced twin challenges of identifying what technology to use and how to design it so that farmers could use the technology effectively. One of the first things Sirajul learned was that the effort of the government to provide agricultural commodity prices to farmers was a failure. The government was collecting information from different markets and posting them on a website. Farmers in most parts of Bangladesh had scant access to the Internet, let alone the ability to navigate through the website to find the information they were interested in.

Bangladesh has a population of 186 million (in 2010) with a population density of 1000 per square kilometre, making it one of the most densely populated countries in the world. Moreover, 72% of the population lives in rural areas, where the agricultural sector constitutes 20% of the total domestic product. About 50% of farmers are said to be below the national poverty line, pegged at an income of USD 1.25 per day. In 2010 the adult literacy rate was 55%.

Sirajul found that there was no clear answer to the question of how information could be disseminated to the farmers. He then decided to conduct an extensive study by which he could ascertain what information farmers wanted, what kind of ICT they were comfortable with and how the information could be communicated to them. He went to 13 (out of a total of 64) districts in Bangladesh, and surveyed over 400 farmers from 50 villages. He also conducted focus group discussions, by getting together a bunch of farmers in a village and having a discussion around their needs. He spoke to government officials, district officials, prominent citizens in the region, local politicians and policy makers, and wholesalers and retailers of agricultural products.

Sirajul found that of all the ICT services, mobile phones have the highest penetration in Bangladesh. Mobile penetration was at 45% (in 2010), as compared to a penetration rate of 0.6% for the Internet. One of the most important findings of the survey was that income levels were not significant in predicting who would own a mobile phone. This meant that even the poorest farmers owned or had access to

CASE STUDY: AMIS for Farmers in Bangladesh

a mobile phone. This demand for a mobile phone was determined by lifestyle issues, as many rural people are young and wanted to own a mobile, as well as the need to communicate to family members residing elsewhere.

Bangladesh villages also have the 'village phone' (VP) ladies, women who rent out phone use. These women are entrepreneurs who acquire a mobile phone and allow residents of their neighbourhood to use the phones for a nominal charge. This has increased access of rural residents immensely. VP ladies have subsequently increased the scope of their services, providing rental access to fax, computers and also the Internet.

Phone users were more comfortable with using the voice features than the text-based SMS service. This discomfort had to do, partly, with the low literacy levels of the population. However, Sirajul's study showed that the levels of usage of text services were increasing, along with a better understanding of how to use the system.

His study also showed that the farmers still relied extensively on relatively primitive modes of farming and transportation. For instance, the bullock or ox-cart and hand-cart were frequently used for transporting goods. Farming relied heavily on manual labour, as opposed to electric-based machinery (electric power supply is very poor in rural Bangladesh). They also relied on price-boards at local agricultural markets to access price information.

Given the state of agriculture and the nature of ICT penetration, Sirajul formulated a design of an Agricultural Market Information System (AMIS). The system relied on the mobile phone as the essential mode of delivery of market information. The detailed design of the system is briefly mentioned below.

The input data to the system would consist of price information related to various commodities and the markets in which they are being sold. Price information collectors would spread to all the different markets and yards and collect price information at different time points during the day. This information they would send back to a central server using text messages. The text messages would be aggregated under different categories and market heads with the help of a software package on the central server. The information would then be disseminated to farmers on a 'push' and a 'pull' basis. Farmers subscribed to the push service by specifying the produce and markets for which they wanted the information and this could be sent to them through SMS. Each broadcast of information would consist of the name of the produce, the highest and lowest prices for the day and the market at which that price was recorded. On the pull system, farmers could text their query to the central server, and then the server would respond with the relevant price information.

The design of the entire system – how data would be collected, how it would be aggregated, how it would be verified and how it would be disseminated – was based on the requirements that Sirajul had gathered from his surveys of all stakeholders. He had to make many design choices, about the manner and details of data collection and dissemination, which were made based on the needs and priorities of the farmers and the possibilities that could be exploited. Although many farmers were aware of and used mobile technologies however they could not articulate, and this was also not expected, how the system had to be configured. They did clearly articulate their needs and the resources available to them to access any information that is available.

Sirajul eventually implemented the AMIS, which was called PalliNET, for farmers and experimented with it for about 6 months. The results showed that farmers were comfortable with the system and used the market price data that was SMSed to them.

# 10.1 ANALYSIS OF BUSINESS PROCESSES

## 10.1.1 What Is a Business Process?

A process in a business is any activity that requires movement of information between individuals, departments or functions. Furthermore, a process is an activity that is of value to the organisation and necessary for it to achieve its goals. A process can be quite complex consisting of many tasks ranging across many individuals and many departments. On the other hand, it can be a simple one involving only one individual sending out a single message. Processes have to be understood in their context and also their rationale.

Processes revolve around the movement of information. For analysis we begin with understanding a process mainly by understanding how information moves, and why this is required. It is important to contrast business processes with other types of processes that do not involve information. For instance, the manufacturing process of mixing chemicals is not a business process. However, the information that the chemicals have been mixed, and its communication to a management information system is a business process.

Many business processes are standard and cut across organisations, whereas other processes are specific to a particular organisation. For example, the process of making a wage payment to an employee is standard across all organisations, whereas receiving a cheque from a customer may not be prevalent in some. It is useful to identify standard business processes, as they can then be supported with information systems that have these processes built in. Many software packages have in-built standard business processes that organisations can use as they are. For other processes, software has to be specifically created.

## 10.1.2 The Business Analyst

The task of understanding and documenting business processes is often done by an Analyst. The business process analyst is often a specialist position in many software and consulting organisations, which is staffed by persons who are trained in using the methods and tools of systems analysis. Their job is to understand processes in the context in which they arise by talking to those who manage and are involved with the processes. They document the processes in a manner that is easily understandable by all and can be used to create designs for systems.

By nature, business processes are often obscure and mired in details. For example, in a government office, to give a clearance for a particular task, the officer in charge may have to consider many binding laws and regulatory conditions. The process of giving the clearance then becomes a matter of what the officer knows of the many cases he/she has seen. After many years, officers and managers often internalise these processes and are not able to specify all of them quickly or in great detail. The job of the analyst then is to work with such officers to understand the full complexity of the situation and the processes involved.

## 10.1.3 Analysis Tools

A number of diagramming techniques are used to understand, analyse and document business processes. These tools are available as software packages. The tools

are used to create graphic representations of business processes whose main aim is to communicate. Since business processes are mainly known to those engaged within them, the purpose of tools is to create a precise document that others can read and understand.

Such tools are part of what are referred to as *structured methodologies*. Structured methodologies are step-by-step ways by which processes are understood, analysed, designed and constructed. As they are structured in nature, it is assured that each step will build on an earlier step, and the design and implementation progresses from concepts and principles to modules and systems. Structured methodologies rely on many tools and proceed in a top-down manner.

Different tools serve to analyse and describe different aspects of business processes. They also focus on different levels of detail. Given below are brief descriptions of some tools and the purpose for which they are used. In Section 10.1.4, one tool, the data flow diagram, is explained in more detail.

### 10.1.3.1 Flow Diagrams

This is a high-level tool that describes processes as a collection of tasks and information flows between them. Flow diagrams are used in Chemical Engineering to draw diagrams of chemical flows in a factory. Another version of flow diagrams is also used in Electrical Engineering to depict flow of electricity. For business processes, flow diagrams may consist of elements to depict tasks, flow of information, information stores, start points and endpoints. Figure 10.1 shows a basic activity flow diagram. The rectangles indicate activities or tasks that are performed, the arrows indicate information flows, the circles indicate data stores, and the small circles show the start and endpoints. The objective of creating such a diagram is to depict the flow of information between activities. It gives a higher level view of what information is used by tasks and what information is produced by them. It also shows what tasks are relevant and have to be included in the analysis, and which ones are not (they will not be shown).

### 10.1.3.2 Flow Charts

One of the earliest methods for depicting logic was the flowchart. This tool was developed to illustrate the logic and flow control of a computer program, also called an algorithm. This tool is widely used to show how a program will use data, decisions and processes to complete a computation.

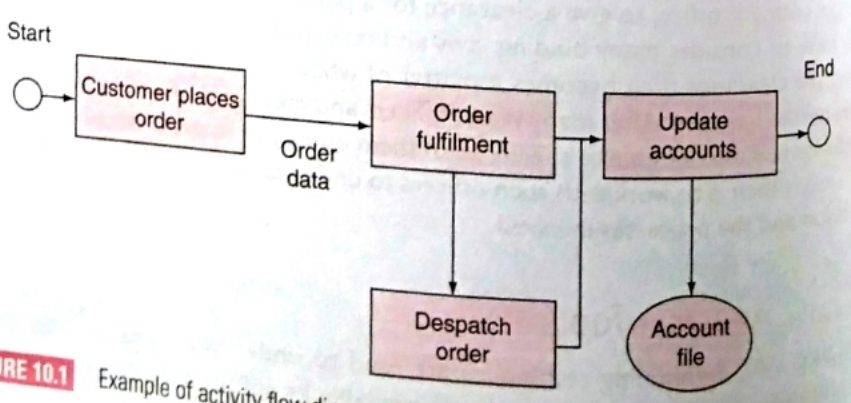



**FIGURE 10.1** Example of activity flow diagram.

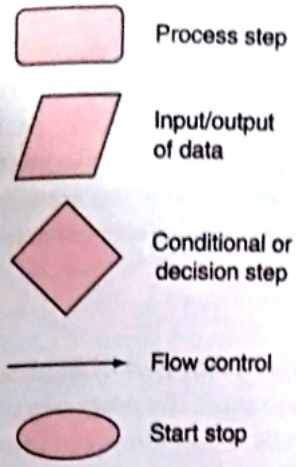



FIGURE 10.2 Flow chart symbols.

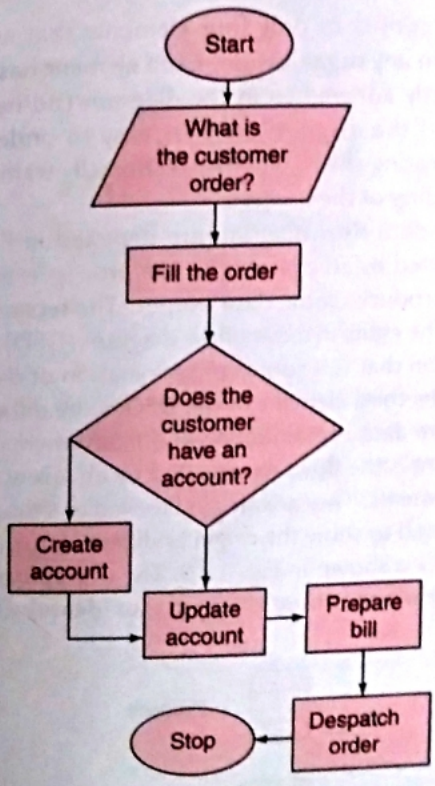



FIGURE 10.3 Example of flowchart.

The elements of the flowchart are shown in Fig. 10.2. They consist of a start/stop symbol, a rectangle to denote a process step or a computation, a diamond shape to represent a decision, a parallelogram to represent data input/storage and arrows to indicate flows. These symbols represent programming steps and control, but flowcharts have been adapted and used to depict business processes also.

Figure 10.3 shows a business process by using a flowchart. The details of the steps are as in the previous example depicted in Fig. 10.1. A customer places an order, the

order is filled and the accounts are updated. And, if the customer is a new one, a new account is created, the bill is prepared, and the goods and bill are despatched. This diagram has a decision point where it is checked whether the customer has an account or not. There are two possibilities at the node, and for each a different route is specified.

Flowcharts do not show all details of information flows. They are appropriate for the low-level depiction of activities and how they are sequenced. They are typically used in conjunction with other tools to specify details of activities and the logic for them.

### 10.1.4 Data Flow Diagrams

Data flow diagrams (DFDs) are widely used by analysts to gain an understanding of the sources of data in an organisation, the processes that modify data, the storage of data and the manner in which data flows between these elements. DFDs are typically drawn in a sequence of hierarchical flows where each diagram, lower in the hierarchy, shows more detail than the earlier one. The details show each process consisting of more detailed processes.

Data flow diagrams consist of only four elements that are used repeatedly to document the processes in any organisation. Each element has a particular meaning and that meaning is strictly adhered to in the diagram (no new interpretations are allowed). The elements of the diagram are thus easy to understand, and all those who are involved with creating them, whether technically trained or not, are able to have a shared understanding of the analysis.

The elements of the data flow diagram are depicted in Fig. 10.4. The first element is the process, depicted by an oval shape. The process is any task or activity that takes data as input and produces some data output. The second element is the entity, depicted by a rectangle. The entity in a data flow diagram (DFD) represents a person or department or organisation that is a source or termination of data. Entities either produce data or receive it. The third element of the DFD is the data store. This represents a file or storage unit where data is maintained and from where data may be read. The fourth and the last element is the flow, represented by an arrow. Arrows show the flow of data from the other elements. They are always labelled to show the type and nature of data. Flows are also directed to show the origin and destination of data.

An example of a DFD is shown in Fig. 10.5. The top part of the figure shows the context diagram. The role of the context diagram is to identify who the entities are and

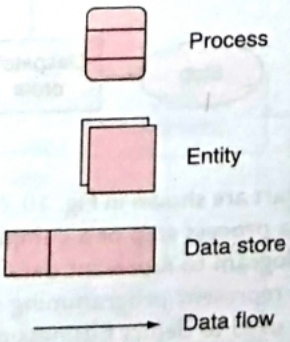



**FIGURE 10.4** Elements of the data flow diagram.

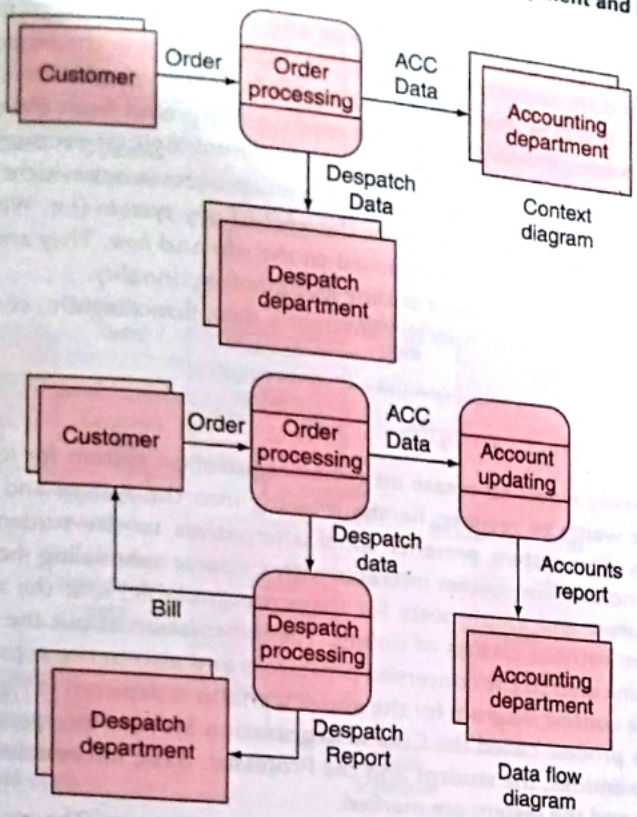



FIGURE 10.5 Example of data flow diagram.

what the main process is through which the entities share data. The context diagram helps circumscribe the analysis, or show the boundaries of the analysis. For the system being considered, other entities in the organisation are not relevant and hence not included. The single process, the order processing, is what is of most relevance. Other processes in the organisation that are not related to order processing are not included in the diagram. The diagram thus clearly shows the scope of the system that is being analysed. The data flows in the context diagram show the most important information that is used within the system. Detailed data flows are shown in later diagrams.

The lower part of the diagram depicts a detailed DFD. The order processing process is now broken into three processes

1. Order processing.
2. Account updating.
3. Despatch processing.

When a customer entity places an order, a data flow labelled order is shown to carry data to the order processing bubble. This process then produces two further data flows – one to the account updating process and another to the despatch processing process. Data carried by the data flows to these other processes is used to trigger further processes. Despatch processing creates a despatch report and a bill that are sent to the despatch department and the customer, respectively. Account updating produces a single report that is sent to the accounting department entity.

There are several points to note about a data flow diagram. First, a data flow diagram does not necessarily show the order of flows or the sequence in which things happen. It simply marks the direction and content of data flows. Whether one process precedes another cannot be read or interpreted from the diagram. Second, a data flow diagram does not depict the internal logic or decision points of any process. How the data is processed inside any process is not evident in the diagram.

Data flow diagrams capture the *what* of any system (i.e. What has to be done? What are the inputs?), as opposed to the *why* and *how*. They are a powerful tool to rapidly scope a system and outline its basic functionality.

As an example of how to construct a data flow diagram, consider the following business scenario.

### 10.1.4.1 DFD Example

A university wants to create an online registration system for its students. When a student wants to register, he/she may log into the system and check for available courses. The system presents some alternatives to the student who then makes some choices. The system interacts with a course scheduling module to ensure that the courses and ample seats for them are available, and the student can register for them without clashes of timing. The information about the student's choices is communicated to the concerned professors and also to the accounting department.

The context diagram for the above scenario is depicted in Fig. 10.6. It consists of a single process called the Course Registration System. Interacting with this process are two entities, the student and the Professor. Basic information flows between the entities and the system are marked.

At the next level is the detailed data flow diagram. The student entity provides information to the Course Registration System regarding courses needed. The Course Registration System passes information to the Scheduling System that provides information to the student about available courses and their timings. A separate data flow from the Scheduling System indicates to the student the courses that have been registered. This system also sends a report to the Professor entity regarding the registrations. The Course Registration System also sends data to the Accounting System that then stores the data in a data store called Student Accounts. The Scheduling System stores data regarding registration for courses in the Courses and schedules data store.

The above example is a simple description of the manner in which data flow diagrams are created from business processes. Here a notable point is that there is no sequencing of data flows and processes. It is important to note what data flows between, say, the student and the Course Registration System, not the sequence and logic by which this flow occurs. Furthermore, details of the three processes depicted in the figure can be expanded in lower level diagrams.

### 10.1.4.2 Some Cautions While Creating DFDs

Data flow diagrams are easy to understand and can be created reasonably quickly. However, to be useful some cautions have to be taken to ensure that the diagrams are accurate.

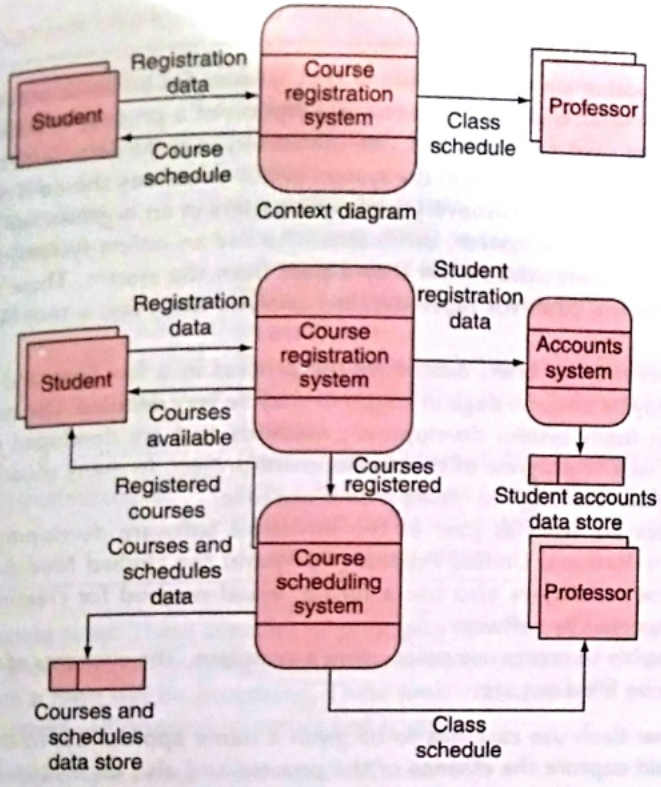



FIGURE 10.6 Course Registration System – data flow diagram.

1. All processes must receive at least one data input and produce at least one data output. Without an input they cannot possibly be functional, and if they do not create an output, either to a store or to an entity or to another process, then their role is questionable.
2. All data stores and entities must either receive or give out at least one data flow. If any data store or entity does not either receive or emit one data flow, their presence in the diagram is questionable.
3. All data flows must originate or terminate at a process. This implies that data flows cannot directly connect entities with other entities, or data stores with other data stores. Furthermore, data flows cannot directly connect entities with data stores. Connecting two entities with a data flow is a very common error while creating data flow diagrams.
4. Entities depict groups, departments or individuals. In the above example, it is correct to denote all students at the university as a single entity. The system is designed for all of them, even when it is known that many of them will have different behaviour patterns with respect to the system. When the entity is a single individual, such as a manager, then the analysts have to be certain that this particular individual is the source or destination of data.

### 10.1.5 Use Cases

Use case is another powerful technique that is used for business process analysis. At a very basic level, a use case is a text description of a process or activity in which some user is involved. Use cases describe the activity in some detail with the objective of clarifying what is needed from the system and in what way should it respond. Use cases are often used as a discovery tool, where users in an organisation, or customers who interact with a system, or citizens who use an online system, are asked to describe in their own words what they expect from the system. These descriptions can then form the basis for more detailed analysis with, say, a tool like data flow diagrams.

Use cases may be brief, describing the process in a few lines and some other details, or may be about a page in length or may be very detailed. Use cases are used extensively in many system development methods and are developed to a level of detail based on the progress of the development project. In many situations, using a simple template is enough to create a brief use case.

Use cases are used as part of the formalisd software development methods known as the Rational Unified Process (RUP) and the Unified Modeling Language (UML). These techniques also use a formal visual method for creating use cases, which is supported by software.

It is possible to create use cases using a template. The elements of the template that have to be filled out are:

1. **Name:** Each use case has to be given a name appropriate to the process. It should capture the essence of the process and also be unique. For example, customer call; citizen request; order fulfilment.
2. **Description:** A brief text description of the use case.
3. **Actor:** Each use case has to involve an actor that is an external agent or entity. The actor initiates the case and also receives some output from the system. Actors are humans, such as citizens or customers, or they are other systems that are external to the system under study.
4. **Precondition:** Each case has to have some preconditions that have to be met. If the precondition is not present or met then the case fails. For example, a precondition for a citizen to reach an online system is access to the Internet.
5. **Trigger:** An event or a time at which a use case is initiated. Triggers are different from preconditions; their occurrence is necessary for the process described in the use case to be initiated, with or without the presence of preconditions.
6. **Scenario:** This is the description of the steps or the sequence of events that happen or are likely to happen when the actor interacts with the system. This is the most likely set of events that are planned for and will be formally built into the system.
7. **Alternate paths:** These are the likely alternatives to the scenarios described above. Alternate paths show the possible ways the process behaviour could proceed in, and the manner in which the system could respond. This is a detailed step often left out of initial use cases.

Table 10.1 Example of a Use Case

| Use Case Name | Process Order Request |
|---|---|
| Description | A customer places an order for a catalogue item on a website. The order is recorded and the customer's credit card is checked for sufficient funds. |
| Actor | Customer |
| Precondition | The customer has selected one or more catalogue items. |
| Scenario | • The customer places an order based on a catalogue item and provides information about his/her account.<br>• The account information is recovered and the customer's credit card details are accessed.<br>• The credit card is verified to see if sufficient funds are available.<br>• If the card has sufficient funds, the order is placed firmly and a despatch request is posted.<br>• The customer is informed of the order details and the expected date of delivery. |
| Alternate scenario | • If the customer does not have an account then a new account is created.<br>• If the credit card is not verified then the customer is asked if he/she would opt for a cash-on-delivery option. |

8. **Business rules:** These are rules or principles by which the organisation is run. These could be accounting rules to make tax calculations, or the sequence in which a form will be processed. These limit and scope the manner in which the system will respond to actors and triggers.

An example of a use case is provided in Table 10.1.

## 10.2 LIFE CYCLE MODELS

When the commercial computer industry was born in the 1950s in North America and Western Europe, customers paid for the hardware and the software was provided along with it. Software was developed using principles of engineering, with an emphasis on design before implementation. As the industry grew, there emerged for-profit companies that would create only software, as partners to the hardware companies. The software companies followed creative practices for building software, as software, unlike hardware, could easily be modified and changed because costs of repeat manufacture were very low and no materials were consumed. The software industry grew immensely, and with it grew the problems of 'spaghetti code' and the need for extensive and scientific testing.

In the 1950s the bulk of the costs of the system purchased by any organisation was the hardware (over 90%), as shown in Fig. 10.7. As software grew and became more complex, along with the ability to deliver more complex solutions to address complicated needs, it also grew as a total cost of the system. By the mid-1970s, software constituted about half of the total cost, and by the mid-1980s it was about 80% of the cost of the system. These increasing costs demanded that the production of software had to be streamlined, and aligned more with hardware production methods.

Large contracts for software, with millions of lines of code, required a formal and structured manner in which the code could be written. This need led to the

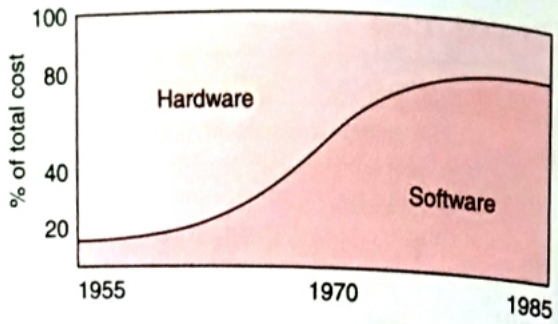



**FIGURE 10.7** Large organisation hardware/software cost trends.
*Source of data: A View of the 20th and 21st Century Software Engineering* by B. Boehm, 25 May 2006.

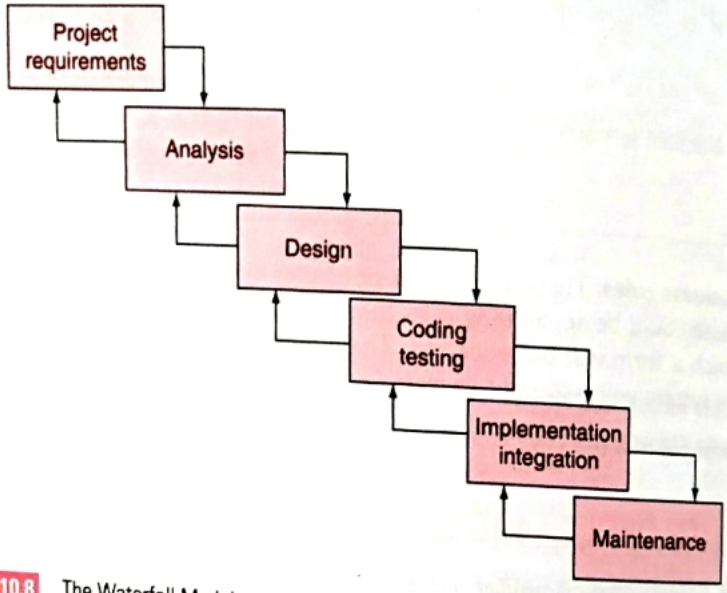



**FIGURE 10.8** The Waterfall Model.

creation of what are known as Life Cycle Models. These models specify a technique by which the coding for the software is preceded by design which is preceded by analysis. Coding is followed by the formal steps of testing and implementation. The first such model was known as the Waterfall Model, depicted in Fig. 10.8.

This model depicts six stages arranged in steps, or a waterfall; the project proceeds along each stage, with the ability to loop back to a previous stage if necessary. Each stage consists of a number of activities that have to be performed and milestones that have to be met. Often the first three are considered the most critical and important parts of the life cycle, called the upper stages of the cycle. The lower stages are more involved with coding and implementation, and though no less important, they are often left to technical management.

The model is called a life cycle as it refers to the entire life of the software, from creation to retirement and replacement by another. Seen in this light many software projects have very long life cycles, spanning decades, whereas many have short cycles, lasting only a few years.

## 10.2.1 Waterfall Model

### 10.2.1.1 Project Requirements

In the first stage of the Waterfall Model, the overall scope of the project is established. The main problem that has to be solved with the system is determined and carefully scoped. Questions that are asked at this stage include: What is the purpose of the proposed system? Who will use the system, which departments, which employees? Will the system interact with other systems in the organisation, and if so, for what purpose? Will external entities such as customers or citizens interact with the system and in what capacity? How long will the system last, what is its expected life cycle? How long will it take to design and implement? How much money has to be allocated for its completion? Who will be responsible for managing its implementation and use?

An important task of this stage is a *feasibility analysis*. This task establishes whether the project is feasible – whether the project is possible at all, given the financial, technical and managerial capabilities of the organisation to go ahead with the project. Once the feasibility is firmly established, it marks a go signal for requirements to proceed further.

The project requirements provide a road map for the project, with important details such as budgets allocated and time frames. Top management has to be involved with this phase of the project, however small or large the project may be in terms of budget or time required. The project team leaders who will have to manage the project through its life have to be directly involved in making and committing to the decisions.

The output from this stage of the life cycle is a document that clearly outlines the scope of the project, the budget, the personnel involved, the likely milestones and the key success indicators. The last head is a set of measures by which the management can estimate how the project is progressing; it includes measures such as person hours on the project, documentation completion, lines of code, team size and budgets spent among others.

### 10.2.1.2 Analysis

This stage is also called the Requirements stage where the basic objective of the system is explained in as much detail as possible. At this stage the most important question that is asked is – What should the system do? To answer this question, it is important to ask potential users of the system what they expect from it. Typical questions at this stage include:

1. What are the main inputs to the system?
2. What will be the various outputs from the system?
3. What kind of subsystems is required?
4. What processes will the system automate?
5. Which processes will be changed to account for the new system's improved performance?
6. What is the expected increase in efficiency?
7. What kind of data will be stored by the system?

8. What kind of reports will be created for external entities?
9. Which other systems of the organisation will the system interact with?

There are many tools by which the analysis is conducted. Some of them such as flow diagrams, data flowcharts, data flow diagrams and use cases have already been explained. Analysts may select one or more of these tools and meet with various stakeholders of the system to record the requirements for the system. Stakeholders are those who are connected to the system in a direct manner, and sometimes indirectly, and whose use or non-use of the system will determine its success. Stakeholders are interviewed by analysts to understand their needs from the system, both currently and in future when they integrate their work with the system.

Analysis also requires working with the documentation and work flows currently existing in the organisation. These documents indicate the data requirements of the organisation, and the work flows are the manner in which data flows. Detailed analysis also involves understanding the subtle and often unspoken methods by which work is accomplished, thus explicating tacit codes in the organisation's work.

The analysis is best conducted by trained personnel, particularly, those who understand the business domain and have a deep knowledge of the tools of analysis. For example, in the banking and finance domain, the best systems analysts are those with knowledge of this sector and who also have deep skills in using analysis tools. Analysts have to use rigorous documentation techniques to ensure that the requirements they unearth are adequately recorded in the requirements document.

The output from this stage is a detailed requirements document that specifies the objectives of the system and what is needed to be done to realise these objectives. After this stage, analysts may feel the need to revise some of their estimates for project budgets and time estimations in the project requirements documents prepared in the first stage.

### 10.2.1.3 Design

Once the analysis is complete and a document containing what has to be done is available, the next step of designing is how the goals have to be achieved. The objective of the Design stage is to provide clear and precise guidelines as to how the system will be built. Details about the logic to be used, the screens to be used, the manner in which decisions will be made and the rules that will be followed are specified. If the analysis outlines what the system should do then the design tells how the system should do this.

The questions asked at the design stage are as follows:

1. How are the data inputs to the system to be collected?
2. What is the nature of the user interface?
3. What are the menu options that will be provided?
4. Where will the data be stored?
5. What kind of data files is required?
6. How will the data be normalised (will it be normalised)?
7. What are business rules used for computations?
8. What is the exact logic for computing certain values?

9. What are the systems, sub-systems and sub-sub-systems that have to be constructed?
10. What will be the architecture for the entire system?
11. How are the different components to be integrated?
12. How much of the system will be off-the-shelf software?
13. How much will be built?
14. How much of the system construction will be outsourced?

A number of tools are used at this stage to specify the design for the system. These tools include flowcharts, structure charts, hierarchy charts, and Computer-Aided Software Engineering (CASE) tools. CASE tools are software environments that provide graphical tools for designing software, and some can also create code on the basis of these designs (see Fig. 10.9). Many programming languages permit quick creation of software prototypes that are used as design tools. A prototype can be created with a CASE tool or with a software programming tool, and it is a working model of the system. The prototype does not have the full functionality of the finished system, but resembles some of the functionality and the look-and-feel that allows users to understand how the software will function eventually and suggest what they would want additionally from it.

The design phase is highly technical in nature, and mainly the technically skilled members of the system-building team are responsible for the final outcomes. They work closely with the analysts and often change the analysis document as more details of the system are fleshed out.

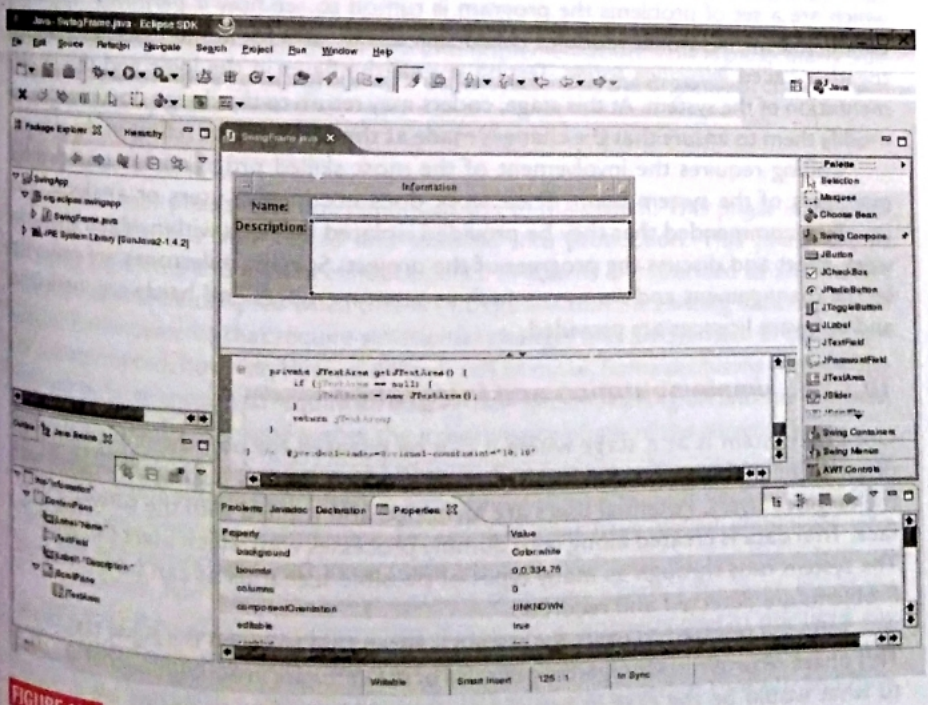



FIGURE 10.9 The Eclipse case tool.

The output of the design stage is a design document that specifies the system in great detail, but does not necessarily contain coded programs. Prototypes may be used for the design but are not a required output from this stage. The idea of the design phase is to de-couple it fully from coding and testing so that the design can be built independently of coding considerations. When the design is complete, it can be handed over to the coding department or to an outside agency to build the system. The design document is thus a blueprint for the entire system.

#### 10.2.1.4 Coding and Testing

This phase involves building the system by writing the software. Coding involves creating software with high-level programming languages. Modern programming languages offer many facilities for reusing pre-built modules through objects. The code is built according to the design specified in the design document.

Some issues that arise at this stage are:

1. What platform (which is a combination of an operating system and programming language tools) will be best suited for the system to be built?
2. What coding and documentation practices will be followed?
3. How will the team structure be constituted?
4. What will be the reporting requirements for the team?
5. If some modules are being outsourced, how will the outsourced part be managed?

An integral part of coding is testing the modules produced. Test suites are created, which are a set of problems the program is run on to see how it performs. Tests are conducted for individual modules called *stub testing*, and for some modules clubbed together called *integration testing*. Testing reveals the flaws in the logic and the implementation of the system. At this stage, coders may return to the design documents and modify them to ensure that the changes made at the coding stage reflect in the design.

Coding requires the involvement of the most skilled programmers and project managers of the system team. Their work does not involve users or analysts. It is usually recommended that they be provided isolated facilities within which they can work, meet and discuss the progress of the project. Specific milestones are reviewed by the management and resources such as team members and hardware, networks and software licences are provided.

#### 10.2.1.5 Implementation and Integration

Once the system is at a stage where it can be released for use, the implementation phase begins. This stage also involves testing, which implies testing the system with the targeted users. Potential users are identified and trained with the software interface. Trial data is created along with dummy processes with which users can interact. The system runs through as many types of possible scenarios as can be anticipated. Problems are detected and recorded.

Software released to users for testing is often said to be on the *Alpha* test phase. This phase returns results regarding the use of the software in conditions that are close to what would be the case in actual use. Reports regarding problems are attended to and the software is revised. After this phase, the software is released to a larger

audience, and this is known as *Beta* testing. Here, the software is in near *production mode*. (A software package is said to be in production mode when it is being used to the full capacity for the purpose it is created for.) Beta testing is more rigorous as it forces the software to encounter situations it has not seen before. This often requires updating the software with patches and fixes.

The testing phase includes integration of the software with the everyday processes of the organisation. After users are trained, the processes of the organisation have to be, usually, modified to accommodate the new software and match the speed and efficiency it introduces. Integration also implies creating new jobs with updated job descriptions and writing new process manuals describing the new methods of working.

Many organisations choose a one-time cutover to the system. The older methods, perhaps using an older software or by manual means, of completing processes is abandoned and the new software is adopted. This has problems as users have to cope with the changes and learn new tools quickly. However, the advantage of a one-time cutover is that it eliminates the old system in one go and there is better adoption of the new system. Another option is the parallel cutover where the old and new systems exist in parallel for a certain period of time, which allows users to both get used to the new system while continuing their work as before, and also gradually migrate their data and files over to the new system. Though this method has its advantages of giving users more time to get used to the new system, it allows some to linger with the old system that may create problems of migration to the new system later.

Usually, large systems take a long time to be tested and made fully operational. At the level of production mode use, more problems will invariably show up that had not been seen or anticipated before. Some of the bugs may be serious in nature requiring an overhaul of both the design and implementation. This is particularly true when the systems have to integrate with other systems in the organisation.

#### 10.2.1.6 Maintenance

The maintenance phase is for the continued use of the system. This phase is initiated once the system has been tested and installed into production. This phase entails recording problems and feature enhancements. Problems are attended to and feature enhancements are attempted when they are possible within the existing structure of the system. Enhancements that require substantial changes and an overhaul of the system are not attempted, however, this is a difficult call to make. Some problems with the system may be large enough to require an entirely new system, kicking off another life cycle, while others may be managed within the maintenance phase of the current cycle.

Maintenance is often outsourced to firms and agencies that have specialist skills in this task. Some systems are often in continuous production for decades after they are developed. They are maintained by agencies that have skilled employees who know the old technology and can create updates and patches for the system to remain relevant. For example, some systems that were created for large insurance agencies with the COBOL programming language, a language that is not used any more to create new programs, about 30 years ago are still in production use. Some outsourcing firms have a pool of programmers who are skilled in COBOL and have taken up the task of maintaining these programs.

### 10.2.2 The Waterfall Model in Perspective

The Waterfall Model and its variations have been used extensively for software development for almost four decades. Standards for the software development process and certifications for the same are based on the Waterfall stages of development.

The Waterfall Model has also become the basis for awarding software development contracts and managing them. The phases of the model are treated as milestones that are monitored and linked to progress reports and payment releases. Government agencies around the world have developed standards for software development projects based on the Waterfall Model. Some inter-country collaboration agreements for development of joint software packages, say for trade purposes, assume the Waterfall Model as the basis for formalising the steps of the agreement.

Despite its wide acceptance, the Waterfall Model also has its share of problems and criticisms. The strongest critique is based on the linear and bureaucratic manner in which the method progresses. In many cases, the feedback aspects of the model are ignored, and the model is treated as consisting of a linear progression of stages, without much feedback to alter the previous stages. Contracts for software development often ignore feedback altogether, treating, say, the analysis phase as completed once the analysis documents are ready, with the implied assumption that the requirements cannot change or be modified in later stages. This has often led to a crisis in the project, where it is realised at the design or implementation stage that certain requirements are not feasible, given the resource constraints of the project. If the requirements are changed then this is called *scope creep* by vendors, and accommodating the new requirements attracts extra charges. In many contracts, scope creep is explicitly accounted for and any requirements changes after the analysis phase are charged a steep penalty price.

The second strong critique of the Waterfall Model is that it slows down the development process, particularly for software that is not very large in scope. With its strong emphasis on analysis followed by design, some developers argue that small software projects suffer from delays. Smaller projects do not need extensive analysis or design; only brief outlines of these are sufficient to go directly to the coding phase. Furthermore, the documentation requirements are also considered to be tedious, particularly for small projects. Many analysis documents run into hundreds of pages for large projects. However, many argue that for small projects these are not required.

When the Waterfall Model is used, the costs of finding and fixing mistakes vary according to the stage at which the mistake was detected and fixed. Figure 10.10 shows the nature of costs incurred for fixing bugs. If an error is introduced at the requirements phase and this is not detected till the maintenance phase then the cost of fixing the bug is the highest. If the error is found at an earlier stage, the cost for rectifying the error is lower. Similarly, if an error is introduced at the design or coding phases, the cost of rectifying it is lower if it is detected in the earlier phases than in the later phases. The figure also shows that the increase in costs for fixing errors increases non-linearly.

The cost of identifying and rectifying errors, as depicted in Fig. 10.10, has two implications. First, the project managers have to ensure that at the analysis and design phases no serious errors are introduced, or extreme care is taken at these phases to

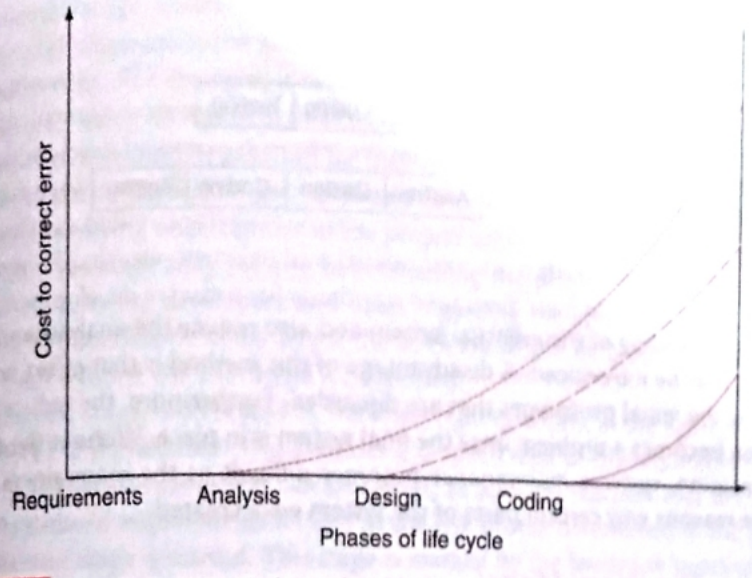



FIGURE 10.10 Cost to correct errors detected at different phases of life cycle.

ensure that the design is near-perfect and can go for coding. This approach is followed by many organisations and for large projects – with close to 500,000 software lines of code. In these cases, almost 50% of the effort within the project is dedicated to analysis and design. Second, the project managers may follow a variation of the traditional Waterfall Model with having short durations of analysis and design followed by coding and testing. The entire system is thus not analysed and designed completely, but is done in manageable parts. This approach has led to a number of alternative approaches to software development, which are described below.

## 10.2.3 Alternatives to the Waterfall Model

### *10.2.3.1 Rapid Prototyping*

Prototyping in software development implies creating a working model of the system, not the entire system. Rapid Prototyping is a formal method by which the system is built in a sequence of cycles of the Waterfall Model. The phases of analysis, design, coding and testing are compressed into a shorter time frame. A partial system is built, and then this is used for analysing more detailed requirements for the next prototype. Detailed documentation is not done. A larger prototype is then built, again going through the phases of analysis, design, coding and testing. The working prototype that is now obtained is larger and more complex than the previous one, and is closer to the eventual system that is desired. The first prototype is now discarded, and the second one becomes the basis for the next round of prototyping (see Fig. 10.11).

The advantage of rapid prototyping is that it allows incremental development of the system, thus, reducing the amount of initial time spent on analysis and design, and giving the systems team a chance to see whether some requirements are feasible or not, and what more can be designed into the system. The prototypes allow for a

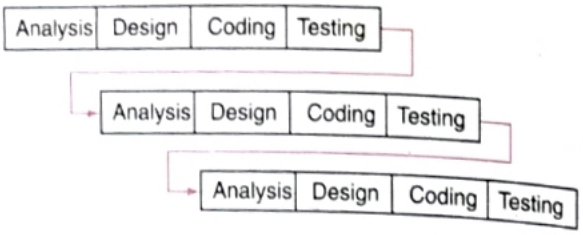



**FIGURE 10.11** Rapid prototyping.

better understanding of the eventual system and also reduce the analysis and design flaws that may be introduced. A disadvantage of this method is that effort is wasted in building the initial prototypes that are discarded. Furthermore, the reduced documentation becomes a problem once the final system is in place. Without the detailed documentation, systems maintenance becomes difficult as the maintainers do not know the reasons why certain parts of the system were created.

### 10.2.3.2 The Spiral Model

The Spiral Model was developed in 1987 to address some of the concerns raised by system developers about the Waterfall Model. The Spiral Model uses an iterative approach, where prototypes are built successively in a spiral of expanding capabilities. It was originally designed for large projects, those that require considerable time and effort to build and would run into millions of lines of code. The model is not an alternative to the Waterfall Model as much as it is an enhancement.

The stages of the Spiral Model are depicted in Fig. 10.12. The model is depicted as a spiral that increases its diameter along the orthogonal axes. The spiral starts in the top left quadrant, and this is the first stage. The spiral is kicked off by a felt need for a system and an assessment of the alternative ways in which the need could be addressed. Objectives of the system are estimated along with estimates of the budget.

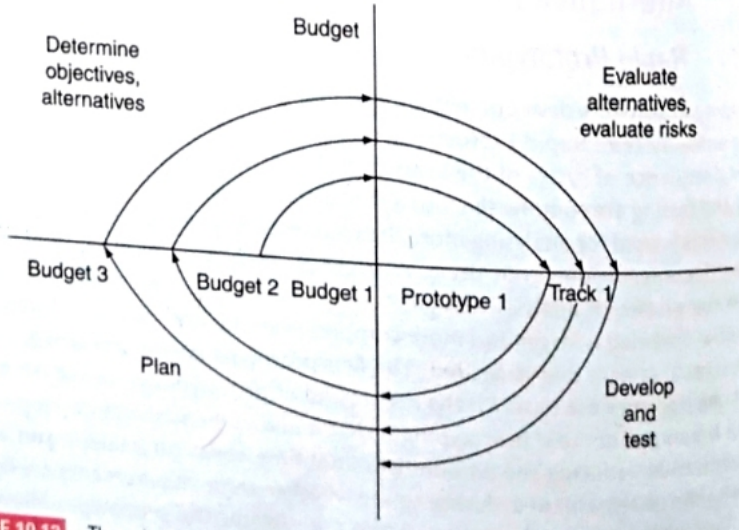



**FIGURE 10.12** The spiral model of software development.

In the next stage, as depicted in the top right quadrant, the alternatives are evaluated. For each alternative, the evaluation involves estimating the risk associated with the development. For example, if the two alternatives for a system for setting up a database are – using a proprietary software or using an open source alternative, then these two choices would be assessed for their risks. Risk assessment involves assigning estimates to the construction, use, value and eventual success of the two alternatives. It involves identifying uncertainties in the project and manner in which this could be dealt with. This stage may include benchmarking the project against other similar projects, interviewing developers and users regarding the effort, and simulating or visualising the conditions under which the software would be used. Once a risk profile has been created, the prototype is developed.

At the next stage, represented by the lower right quadrant in the figure, the prototype is tested and evaluated. This provides a detailed idea of the requirements of the system, now that the prototype can be tested, as also the features that are missing. Using the detailed requirements as well as the risk profile determined in the previous stage, the next stage is started. This stage is marked by the lower left quadrant in the figure. Here plans for the next iteration in the spiral are made. These plans are based on the lessons learned from the first prototype and its evaluation. The plan includes the life cycle of the product, including all stages of analysis, design, testing and coding.

As the next cycle of the spiral is initiated, the activities of setting objectives, determining alternatives and setting a budget are initiated again, followed by the next stage of determining the risks for the alternatives. A new prototype is built, followed by its evaluation and a new set of plans for the next iteration.

The Spiral Model of iterations ends when the risk analysis shows that the system is well enough understood to go into the final round of analysis, design, coding and testing. The requirements are known well enough now, and so the focus is on the design of the eventual system and the testing.

The Spiral Model is a risk-driven model that explicitly models risk and uncertainty in the development process. The entire focus is on understanding and controlling risks to reduce the chances of eventual failure of the system.

### 10.2.4 Agile Methods

The late 1990s saw the evolution of software development methods known collectively as the *Agile Methods*. Drawing inspiration from manufacturing quality processes, these methods relied on short analysis-design-coding-testing cycles, with close user interaction and flexible team compositions to build software. The underlying emphasis in the Agile methods is on reducing and controlling risk. Shorter life cycles ensured that bugs and design flaws were detected early, and the user involvement throughout the project ensured the requirements were addressed adequately.

The Agile manifesto, a document outlining the beliefs of the developers who invented the Agile techniques, defines the fundamental propositions of these methods. The Agile methods rely on creating programs quickly and with close interaction with the customer. Requirements are never frozen and change is welcomed to ensure that the customers' needs are attended to right through the end of the project. Developers are given autonomy and responsibility to work on their own and seek out

quality measures for the project. The Agile methods also require suitable working conditions like an open working environment and a limit on the number of hours that can be worked on in a week so that developers remain motivated and productive.

Different agile methods have different means by which they address the objectives of the manifesto. Two of the methods that have gained popularity are Extreme Programming and Scrum. These methods are described below.

### 10.2.4.1 Extreme Programming

Extreme Programming has gained wide popularity among the Agile methods. It embodies most of the objectives of the Agile manifesto and has been demonstrated to be effective for small- to medium-scale projects. As a method, Extreme Programming has a few core ideas.

1. **Stories:** All activities are initiated by stories that users tell about what they want from the system, how they go about doing their work and what processes they follow among others. These stories are then written on index cards and the writing is done rapidly. The stories constitute the initial analysis for the system.
2. **Short releases:** Modules for the software are released quickly and often. A module is determined by the users and the developers jointly, based on the highest priority of features wanted. Here, users have a priority as to what they want to see first, and the developers guide this choice based on cost and resources available. The releases are made as frequently as every day.
3. **Testing:** Releases of code are tested as soon as they are created. In fact, while planning for the module, the test sets are created and a module is considered completed after the tests are satisfied. This manner of testing is known as *unit* testing.
4. **Iteration:** The iterations in Extreme Programming are based on the features needed in the releases. The system is put into production the moment the release is made, or it is placed in the central repository (a repository is a central collection of code). The units are integrated and the whole package is tested. If it fails then the latest changes are reversed. If the changes are accepted then this starts a new iteration with another story and another set of features.
5. **Refactoring:** The design of the system evolves and changes as the system is built up from the modules. Refactoring, or changes, is done as the integrated system needs them.
6. **Pair programming:** All the coding is done by a pair of developers. One writes the code and the other 'looks over his/her shoulder' to see how the code fits in with the rest of the system and how it can be thoroughly tested. Programmers have to be highly skilled in doing their task and they have to work rapidly.
7. **Work environment:** The entire team is located in one room, with the programmer pairs at a centre table/computer. At least one customer is always available for evolving the features and to write the stories. In total, 40-h work weeks are desired. However, occasional overtime is permitted, but not for successive weeks.
8. **Metaphors:** Both the developers and the users work with metaphors to visualise their system. Metaphors are words or phrases that resemble some desired

feature or attribute, but are not exact. Metaphors help seek out different meanings that the users may have about what they want, and explore different possible ways of achieving the same.

Extreme Programming is successful because it allows the developers autonomy in understanding the problem clearly and designing the system accordingly. The close interaction with the customers also enables continuous dialogue and an understanding of implicit and tacit codes that are often missed in formal meetings. Furthermore, its emphasis on continuous testing and integration ensures that the evolving system is always working and meets the customers' needs.

### 10.2.4.2 Scrum

The Scrum method relies on the idea of evolving the requirements for the project and the needed activities on the basis of work completed rather than on plans. The project proceeds in a series of *sprints*, which are fixed durations of 2 or 4 weeks. In each sprint particular and defined objectives are met; a tested system is built for that portion alone, and what further needs to be done is determined on the basis of the completed work.

Scrum involves defining a number of roles for the project members and the creation of a number of artefacts.

1. **Product owner:** He/she is a customer or client who determines what the system should be and how it should behave. He/she is a part of the team and is always involved.
2. **ScrumMaster:** He/she is a leader for the Scrum team, who acts as a project coordinator.
3. **Team member:** He/she is a member of the team, who participates in building the software.
4. **Product backlog and sprint backlog:** These are requirement documents that state what has to be achieved for the project and for each sprint. Each backlog is essentially a high-level requirement specification that is fleshed out and detailed during the sprints.
5. **Burn down chart:** This is a record of backlogs that have been completed. For a sprint, the burn down chart shows what has been completed on a daily basis, giving the team a focus for what needs to be achieved.

During a sprint, meetings to review the progress and create the burn down logs are highly structured and controlled (see Fig. 10.13). A daily meeting is held for exactly 15 min in the same place and at the same time (first thing in the morning). All may attend the meeting but only the 'pigs' are permitted to speak. Pigs are the product owner, the ScrumMaster, and the team members, or all those who are committed to the project; commitment implies they are accountable to the progress of the project. Others are not permitted to speak, and are termed 'chicken' as they are only involved with the project but do not have a commitment. Meetings determine what has been achieved during the previous day's sprint and what will be achieved today. The ScrumMaster determines if there are any obstructions to the progress, and if so, these are resolved. After this meeting, the team members meet to discuss various areas in which collaboration is required.

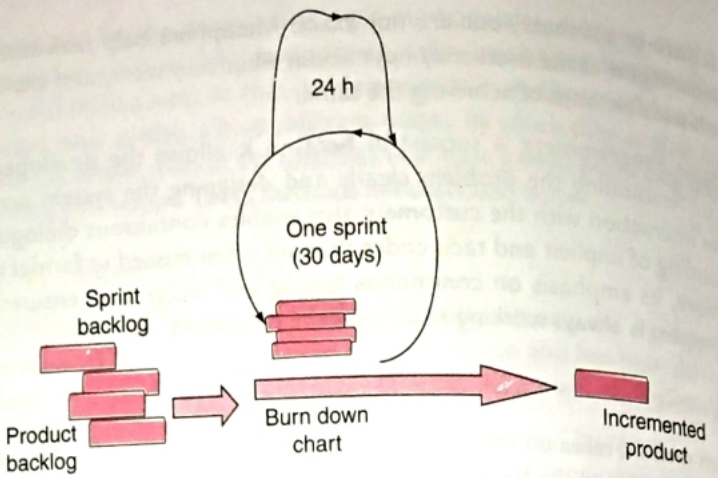



**FIGURE 10.13** The scrum process.

Scrum meetings are also held after each sprint to establish a new set of backlogs based on what has already been achieved. A Scrum ends after it is realised that the system has met all the objectives stated by the product owner. Each sprint produces a working version of the software, albeit incompletely. As the sprints progress, the software grows and begins to meet all the requirements.

The difference between Scrum and Extreme Programming lies mainly in the diffuse manner in which Scrum proceeds. Requirements are specified at the high level and get focused and tightened only as the development progresses. Furthermore, requirement changes are considered to be inherent in the product evolution process, and after each sprint changes can be quite significant.

## 10.3 SOFTWARE PROJECT MANAGEMENT

The frameworks or methodologies for software development present a structured manner in which software can be constructed. However, the frameworks do not include a large number of issues that arise as a project unfolds. Some of these issues are common to projects of all kinds, whether for software development or for some other activity, and some are widely different. For software project management, the most substantial issues have to do with finding requirements, managing a team, ensuring the final product works, and trying to stay within budgets of time and money.

The project management issues arise in all contexts of software development, regardless of the particular methodology followed.

### 10.3.1 Top Management Support

Possibly one of the most important issues for software project management in the current organisational environment is that of obtaining and retaining top management support. Top management buy-in to the project is required from the start, even while the project is being conceptualised. The requisite resources have to be allocated and then supported for the duration of the project.

The distinct advantages of having the top management involved are many. Large and medium-sized projects, in particular, require integration with other systems within the organisation, which the top management can facilitate. The vision and high-level requirements for the software are provided with assistance from the top management, ensuring that the software integrates well with the larger goals of the organisation. Vendor involvement and selection are facilitated with negotiations at the highest levels of the organisation. Also, for the organisation to accept the new system, the top management is indispensable for providing the motive and rationale for the new system, ensuring personnel training and reassignments.

## 10.3.2 Team Composition

The team that is involved with the overall design, implementation and maintenance of the system has to be constituted carefully. Teams have to have a balance of skills and experience in project management. Ideally, teams should last the duration of the entire project cycle, however long the cycle may be. Rules for team composition are hard to define, as they are so dependent on context. However, some avoidable issues can be highlighted as they are known to be serious problems.

1. The project leader should be an individual who commits to the entire duration of the project. A replacement should be clearly identified and groomed, in case the leader has to be away for unavoidable circumstances.
2. Skills of team members should be carefully documented and made known. This will ensure that others know who can do what, and so they can call upon them when needed.
3. Team members should be comfortable working with each other. More importantly, if there are known problems with individuals, they should be addressed early in the project by the management.
4. The leader should have overall accountability and responsibility for the project. He/she should be the person one is to turn to if things are not going well. The leader should also be the person to be held responsible if the project fails.

In modern, large organisations, it is often the case that teams are distributed across the globe and work jointly using electronic networks for communication. System building assumes a more difficult form in such situations, although the above critical concerns have to be addressed.

## 10.3.3 Vendor Selection

Organisations nowadays use vendors extensively to build and help maintain their systems. Vendors are often hard to identify, particularly those with special skills. Industry associations provide directories that list software vendors and their skills. In India, the National Association of Software and Services Companies (NASSCOM) is a well-known association that maintains a database of vendors. This is a good

starting point. Most client firms will make it a point to evaluate a vendor thoroughly before they are selected. Some points for evaluation are:

1. The number of projects the vendor has completed of a similar nature in the past. This should also include the size of the projects, the time taken for completion, the kind of technology used and whether the clients were satisfied with the project.
2. The skills base of the vendor depends on: (a) How many employees they have with particular skills, (b) how many have been with the firm for long, (c) what has been the training imparted to them and (d) how have they gained experience.
3. A site visit to the vendor's premises to see the working conditions as well as the assets the vendor has to use for the work. This would indicate how deeply the vendor can commit to the project.
4. For long-duration and large projects, the financial position of the vendor should also be evaluated. Some firms demand bank guarantees from vendors against successful completion of the project. Only financially sound vendors are able to produce such guarantees. As small vendors cannot produce such guarantees, the client organisation has to rely on financial records to assess the vendor.

It is also important that tenders and call-for-proposals floated to ask for bids from vendors should not be biased to favour one or the other vendor. The wording of the tenders has to be vendor-neutral, particularly avoiding words or phrases that identify with a particular vendor.

### 10.3.4 Project Estimation

At the initiation phase of projects, one of the most difficult problems managers face is that of estimating the size of a project. This entails estimating the resources that will be required by the project and the effort involved. The entire planning for the project is based on these estimates and if they are wrong, the project success is jeopardised.

Project estimates can be over or under. If they are overestimated, it means that the project will eventually take less resources than planned for. If underestimated the projects will take more resources than planned for. Some argue that it is better to overestimate than to underestimate, as excess resources are not a problem but less resources are. For vendors who are bidding for a project, overestimation leads to a high cost estimate for the project thus making their bid potentially uncompetitive. If their bid is an underestimate, which they also win then they will incur higher costs for the project than they had bid for, thus cutting into their profits. For client organisations, overestimation leads to committing resources more than required, thus leading to higher costs and management problems; underestimation leads to scarce resources that have to be procured, possibly at a high expense.

### 10.3.5 Function Point Analysis

This is a technique to estimate the size and effort required to build an information system. The estimate is given by one measure, called the *function point*, which shows how complex the project is. For example, a project with a function point of 200 can be estimated to be built in a certain number of person hours. This estimate is independent

of the platform and software tools used. The function point estimates are based on two broad categories of computing requirements:

1. **Data functions:** It indicate the variety of data input and output needs and how they have to be built.
2. **Transaction functions:** It indicate the processing required within the system. Transactions are categorised into various sorts such as communication, manipulation, writing, etc., and the efforts required for these are then estimated based on some industry benchmarks.

Function point analysis has been shown to be quite effective in providing an estimate of the size and complexity of the system. However, it cannot account for project-based specificities such as changes in the project scope and requirements, skill levels of developers, availability of platforms and so on.

## Chapter Glossary

Business process Any business activity that involves movement of information between people, groups or departments.

Business analyst A person who has the responsibility of understanding and documenting business processes.

Flow diagrams A high-level illustration of tasks and the movement of data.

Flowcharts A diagram that depicts the logic and flow control of a computer program.

Data flow diagrams A high-level diagram that illustrates data flows between people, processes and data storage.

Use cases A technique for conducting business process analysis that involves descriptions of tasks, people and information that flows between them.

Prototype A scaled down version of a software program that is used for analysis of business processes.

Stub testing A way of testing software that involves examining how responses from independent modules will impact the complete package.

Beta test Testing of software that is nearly complete, has all the features and can be conducted under conditions similar to the real environment for which it has been created.

Parallel cutover Releasing a new software in an environment where the old methods of doing tasks are also kept alive.

Scope creep Increase in the requirements of a software module, much after the requirements phase is over.

Unit testing A method of software testing that involves testing independent modules extensively.

## Review Questions

1. What is a business process? Give two examples of business processes.
2. Review the elements of a flowchart. Then draw a simple flowchart for the following problem: A customer sends an e-mail requesting some information. If the request is about a product, send a product brochure to the customer by e-mail. If the request is for something else, refer to the request to a representative by forwarding the e-mail.
3. Review the elements of a data flow diagram. How are they different from a Flowchart?
4. Draw a data flow diagram for the following problem:

   A system at a library, called the Library Information System, allows a user to check out a book.

When the user requests to borrow a book, the book details are provided to the system, along with the user's details. The date of the checkout and the due date are printed out and given to the user. The system then stores all the details in a file.

5. When are use cases applied? What are some basic elements of a Use Case?
6. Why has software engineering assumed importance in the modern context?
7. Identify the stages of the Waterfall Model and describe their main goals?
8. Describe some limitations of the Waterfall Model?
9. What is Rapid Prototyping and how is it an improvement over the Waterfall Model?
10. How are agile methods useful? Why are they preferred over the Waterfall Model?

## Research Questions

1. What are the different kinds of tools used for software development using the Waterfall Model? Some are mentioned in the text, can you find more?
2. Identify some weaknesses with the agile methods? Are they suitable for all types of projects? Are they suitable for all types of firms?
3. Visit a library in your institution (or in your neighbourhood). If they use an information system for lending out books - try to draw DFDs to analyse their lending processes. Try to capture as much detail as possible. If the library does not have an information system, then sit with the librarian to find out what kind of system would they need and draw DFDs for the same.
4. Search the Internet or in your library for information on systems development projects. How many of these were successful and how many failed? How is project failure measured?

## Further Reading

1. Gomaa, H. (2011) *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, Cambridge University Press, Cambridge.
2. Beck, M. (1999) Embracing Change with Extreme Programming, *IEEE Computer*, **32**(10).
3. Boehm, B. (2006) A View of 20th and 21st Century Software Engineering. Proceedings of the 28th International Conference on Software Engineering, ACM New York, USA.
4. Nerur, S., Mahapatra, R. and Mangalaraj, G. (2005) Challenges of Migrating to Agile Methods. *Communications of the ACM*, **48**(5), 72–78.
5. Islam, M.S. and Grönlund, Å. (2011). Factors influencing the adoption of mobile phones among the farmers in Bangladesh: Theories and practices, *International Journal on Advances in ICT for Emerging Regions*, **04**(01), 3–13.
6. Islam, M.S. and Grönlund, Å. (2011). Bangladesh calling: Farmers' technology use practices as a driver for development. *Journal of Information Technology for Development*, **17**(2), 95–111.