Systems Analysis and Design

Howard Gould

HOWARD GOULD

SYSTEMS ANALYSIS AND DESIGN

CONTENTS

ACKNOWLEDGEMENTS

Trademarks

Some of the product and company names used in this book have been used for the purpose of identification only and may be trademarks or registered trademarks of their respective manufacturers and sellers.

PRINCE2® is a registered trademark of AXELOS Limited.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

SSADM is a registered trademark of the Office of Government Commerce (OGC) an office of the UK treasury.

Unified Modelling Language and **UML** are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.

FOREWORD

This book has been written to provide a concise introduction to systems analysis and design for students studying computing, IT or business related courses. Similarly, others who need to work with systems analysts, designers or software developers when commissioning or whilst using a new information system may benefit from an understanding of this content.

The information is presented in a form that makes it easy to grasp the essential principles and techniques and to apply these within an information system development project.

Contents cover the full system development life cycle and include systems analysis using structured analysis techniques and object modelling with the unified modelling language (UML). The newer agile approaches to systems development are also introduced. Also included is system design, incorporating data design, human computer interaction and system architectures, along with coverage of system implementation and maintenance. A brief introduction to IT project management techniques is also included as an appendix.

Further supporting materials can be found at the author's website http://howard-gould.co.uk/.

1 INTRODUCTION TO SYSTEMS ANALYSIS AND DESIGN

On completion of this chapter you should be able to:

- identify the components of an information system
- understand the purpose of systems analysis
- be aware of the role of the systems analyst
- understand the systems development life cycle.

Information technology (**IT**) based **information systems (IS)** are essential to all types of organisations and in order for these systems to be of benefit they must be based on well-defined requirements and designed and built using **systems analysis and design** (**SAD**) processes.

1.1 WHAT IS AN INFORMATION SYSTEM?

An information system can be defined as a set of interrelated components that function to provide required information for a specified purpose.

A system receives input data which is processed, resulting in meaningful information – output. (In some cases this output may be data to be used as input to another system). An information system, shown in Fig 1.1 below, will have a control mechanism which regulates the inputs and processes based on feedback from the outputs. A thermostat which regulates a heating system is a good example of this. Systems work within boundaries and operate within an environment. Large systems may comprise a number of sub-systems which work together to support the overall function of the main system.
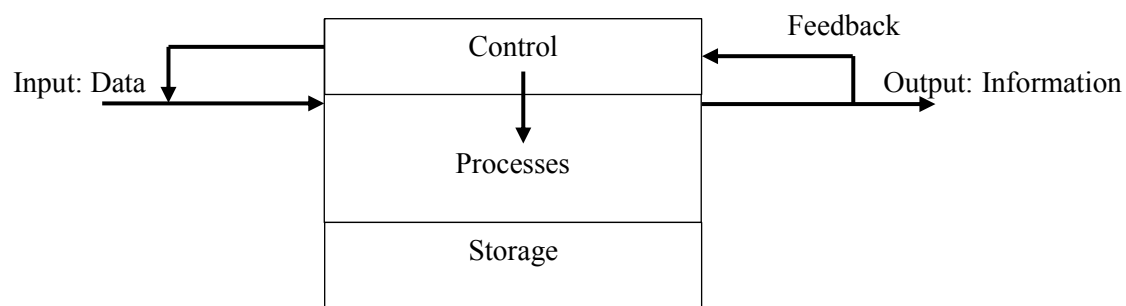
Fig 1.1 Elements of an Information System

The following diagram illustrates the basic elements of a payroll system. In this example, typical inputs would include an employee's hours worked and their tax code. This data, along with previously stored data such as pay rates and employee details, would be used within system processes such as "Calculate Tax" and "Calculate Pay" in order to calculate the employees' pay. Some of these calculated pay details would also be used as feedback which will be used to influence future calculations for tax etc.

Here the system boundary relates to the business and its employees using the system. However, the environment it operates within includes outside agencies such as the tax agency.
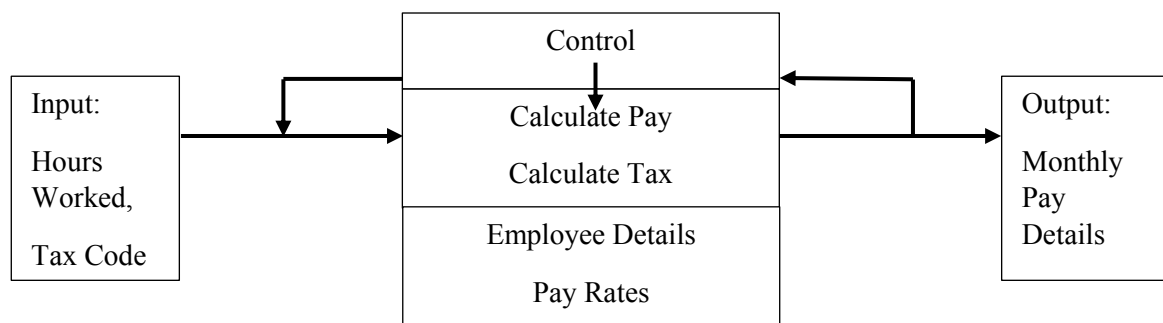
Fig 1.2 Example Payroll System

The main components of an information system are:-

Input – the data that comes from the environment that the system will be operating in.

Process – operates on the inputs and transforms them into outputs, e.g. an application for a driving licence results in a driving licence being issued following processing.

Storage – for data/information.

Output – the results of a process.

Feedback – an output that is fed back into the system to alter the control of the processes.

Control – regulates the system using the 'feedback' to ensure the system operates to meet its purpose.

Boundary – the limit or scope of the system.

Environment – the area in which the system operates. This may include other systems or other organisations.

Hardware – the computers and other data input and output devices.

Software – Programs, operating systems.

Data – input values and information output.

Communication – networking infrastructure, the Internet.

People – system analysts, software developers, system users.

A number of stakeholders are involved in the system development process, including:

> System requester – usually a client or senior manager
> System user – person who will use the system
> System analyst – person who analyses and designs information systems
> System developer – person who designs, produces and tests software.

1.2   THE SYSTEM DEVELOPMENT LIFE CYCLE

The traditional approach to acquiring an information system is to use structured analysis processes within the **system development life cycle** (**SDLC**). Structured analysis relies on process models which show how data flows into a system and is processed by applying business rules, which in turn leads to data being output as required information. The **systems analyst (SA)** plays a leading role throughout the systems development process. A systems analyst needs good analytical skills in order to identify problems and consider effective solutions, and should also have good communication and interpersonal skills in order to communicate effectively with a wide range of stakeholders ranging from senior managers to operational employees (system users).

In recent times newer development approaches have become more team-based, utilising both IT staff and system users to help speed up the process and so cut costs. **Joint application design (JAD)** involves a group of users meeting intensively with systems analysts to help with the information gathering and the system requirements definition process. Similarly, **rapid application development (RAD)** (Martin, 1991) aims to speed up the SDLC, with users being involved in the design and development tasks in a more interactive, iterative way, which means that users can give feedback much sooner on the developed system than in the traditional SDLC.

A newer alternative to structured analysis is **object-oriented analysis and design** (**OOAD**) which focusses on identifying real-world **objects** used in a system; these objects combine business processes with the data that they use. The O-O approach tends to be adopted when a system is going to be developed using O-O programming languages.

The traditional systems development life cycle is primarily a sequential process which is often referred to as a **waterfall model** (Royce, 1970) as it is based on a plan formed at the start of the project, whereby each phase of the SDLC is allocated a period of time to complete and then the outputs are fed through to the next phase until the system is completed. Appendix C gives an overview of project planning.

The SDLC has five main phases, as shown in the diagram below:

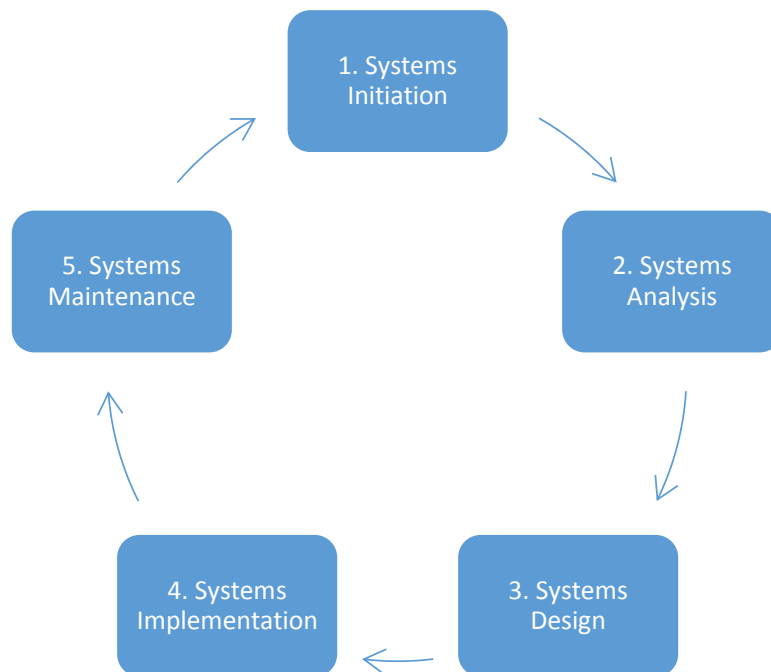**Fig 1.3** Traditional Systems Development Life Cycle

Note. There are a number of variations of the SDLC, with some showing more phases and some including iteration of a phase with the previous one, however the same activities are included in all.

More recently **Agile approaches** (Beck K., et al, 2001) to systems development have become established as a flexible alternative to the traditional SDLC. They follow an incremental/iterative construction approach whereby small software prototypes are enhanced a number of times following regular review meetings with stakeholders (called "scrums", named after the game of rugby) until it becomes an acceptable product. The iterative construction phase includes design, code and test activities and usually has a short time period of one to four weeks called "sprints". The Agile approach aims to produce working software as soon as possible whilst allowing a more adaptive form of planning which caters for changes to system requirements. (Ambler, 2009–12) provides a more detailed explanation of Agile Life Cycles.
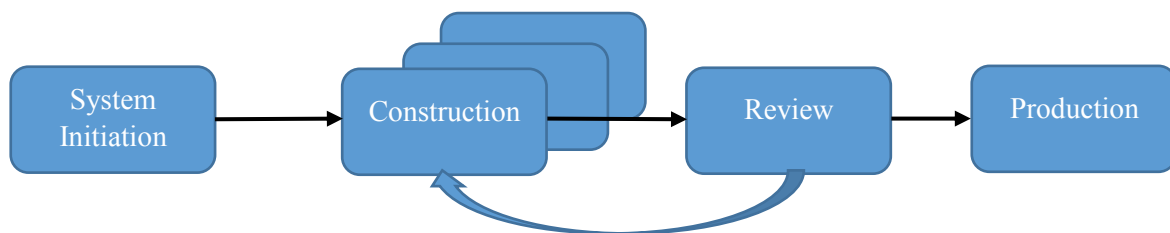
Fig 1.4 A simple Agile Development Life Cycle

The following is a brief summary of the phases of the traditional SDLC:

Systems Initiation

The systems initiation phase is the starting point for a new system and is invoked following a **system request**, backed by a business case which gives reasons for the request; this usually comes from a senior manager or client. The system request is a formal document which outlines the main requirements for a new system, or changes to an existing one. The system request is often driven by the need for IT systems which help the organisation to meet its strategic objectives. These generally relate to a wish to improve the effectiveness of the existing business systems by means of offering better service or performance, or to the need to comply with new external demands, e.g. government legislation. The system request is often produced by the organisation's IT department or an outside consultancy following consultation with the senior managers, clients and other key stakeholders.

Once a system request is received by the IT department (or, depending on the organisation and its resources, possibly an outside software consultancy) an **initial investigation** is undertaken to see whether the requirements can be satisfied by an IT solution. This is usually undertaken by a systems analyst and involves undertaking a **feasibility study** to see whether in fact the project is viable, as there are a number of factors that need to be considered before approval is given. These include whether the requirements can be satisfied economically e.g. within the proposed budgets (development and operational).

A **cost benefit analysis** (Appendix A) is usually undertaken to ensure that the proposed system would be cost effective, though in some cases the benefits of a new system may stem from intangible benefits that are not easy to cost. Technical and organisational restrictions need to be assessed, which includes checking that the proposed system can handle the expected volumes of data and user numbers, whilst maintaining acceptable performance levels. There also needs to be adequate availability of human resources such as development staff and technical resources. Ethical consideration is also given to ensure that the system will be legally, commercially and socially acceptable.

Following the initial investigation, the analyst may decide that the system requirements can be satisfied by making better use of the existing IT systems or altering existing business procedures, thus negating the need for a new system. If, however, a new system is deemed necessary and feasible then senior management will give approval for the system development process to proceed. A more detailed analysis of requirements will be undertaken during phase 2 of the SDLC – Systems Analysis.

Download free eBooks at bookboon.com

Click on the ad to read more

Note. The feasibility study may involve considering a number of alternative approaches to satisfying the requirements. Each feasible approach will be included within the feasibility report. The analyst will indicate the advantages and disadvantages of each approach and make a recommendation, though ultimately the decision as to which approach will be selected will be made by the senior management or client.

Systems Analysis

The systems analysis phase is undertaken by one or more systems analysts and is used to identify the detailed system requirements in order to produce a **requirements specification**. This specifies what needs to be included in the new system to meet the system users' requirements. In order to develop the requirements specification, requirements capture and modelling activities are undertaken. These involve identifying what data is needed by the system (inputs/outputs) and the processes (business rules) which are needed to process the data and produce the required information outputs. Additionally, any performance and security requirements will also be identified.

Note. In the early days of IT systems development, a system was specified by the analyst in the form of lengthy textual descriptions. These were often misinterpreted by developers and users, frequently resulting in systems being produced that did not meet the users' requirements. As the users often did not see the system until it was complete, any misinterpretation of the users' requirements proved costly to rectify. Current system modelling methods incorporate diagrams, which helps to reduce the amount of text and potential for misinterpretation – a case of a picture being worth a thousand words.

This information is arrived at following fact-finding activities undertaken by the analyst, including meetings with the relevant stakeholders of the new system to establish what they require. Other fact-finding techniques include questionnaires, observation of existing business processes and reviewing existing system documentation, including collecting sample system documents. The meetings may take place on an individual basis or during a JAD session with a number of people present.

Where a new system is replacing an existing system, the analyst will initially observe and model the existing system to help gain a better understanding of the users' needs and any existing problems. The analyst will record all relevant information, as this will be referenced throughout the SDLC. This information is usually stored **within a computer aided software engineering (CASE)** tool as these allow the analysts and other stakeholders to access the important system requirements and design information quickly and easily. Some CASE tools can also use this model information to automatically produce software elements needed for the new system.

The requirements specification is used to build a **logical model** which will represent what needs to be produced (but not how). In the case of structured analysis this will include process and data modelling which includes **data flow diagrams** (**DFDs**) along with business process descriptions.

Note. An alternative modelling toolset is the **OMG's unified modelling language (UML®)** which is a flexible modelling language that has become established as the norm for O-O development projects. It has 13 diagram types, although typically only a few will be produced for a given system.

Many organisations do not enforce a modelling methodology rigidly so analysts/developers are often free to choose appropriate models/diagrams for the task in hand. The aim should be to ensure that whatever modelling methods are undertaken they will be understood by those who will be using them.

Systems Design

The systems design phase is used to specify in detail the system to be built, based on the requirements specification identified in the systems analysis phase, so that the system developers are clear about what they need to produce. This phase involves producing physical process diagrams (DFDs) that show how the data will be processed in the new system. The business rules for the processes will also be specified and these may include descriptive text, pseudo-code (written using structured English), decision trees and decision tables. In addition, screen and report layouts will be included to show how the required information will be input to and output by the system.

In most systems there will also be a need to store data within a database so at this stage an **entity relationship model (ERM)** will be produced which will be used to identify the database tables and their data items. This information forms the data dictionary which is usually stored within a CASE tool. Although much of this work will be undertaken by the systems analyst, some will be in consultation with system stakeholders to ensure that the proposed design will satisfy their requirements.

When the design has been completed, the developers will work from the design specification and commence the development work, coding and testing the software. The choice of programming language, e.g. Java, C++ etc. used to develop the code will depend on the type of application being developed and other factors such as available expertise and conformity to the organisation's development environment standards.

When software is developed it needs to be thoroughly tested to ensure that it performs as expected and satisfies the specified requirements. Initial testing will relate to specific modules or programs; when these have been satisfactorily completed integration testing can take place, which involves testing all the modules and programs working together, as they will do when the complete system is implemented. During this phase, if new hardware or networking is required to support the system this will be acquired and tested to ensure that the system will function appropriately when in use. Finally, acceptance testing will take place. This involves asking intended users to trial the system to ensure that they are satisfied before it is released for use. It may be necessary to make some minor amendments at this stage, but if the earlier analysis and design phases have been carried out accurately there should be no need for any major changes.

Note. In some cases, the new system may be implemented using an existing product or package which is purchased or acquired from an external source or vendor. This may require some customisation, so will also need to undergo testing to ensure it functions in accordance with requirements.

Systems Implementation

Following acceptance testing the system will be implemented (handed over for use) in its intended environment. In order to prepare the users, it may be necessary for training on the new system to take place first. An implementation strategy will be specified which will indicate whether the new system will operate in parallel with the existing system for a period of time in case there are unforeseen problems with the new system, though in the case of a completely new system this is not possible. In some situations, it is not feasible to run duplicate systems, so where possible, the new system is rolled out in stages, in order to minimise risk.

Systems Maintenance

Once the new system has been implemented and is in normal use it is monitored, and any minor problems or bugs (errors) dealt with as soon as possible. Reviews will also be scheduled to ensure that the system is performing as expected. If any non-essential additional requirements or changes are requested these will be considered and scheduled for implementation if approved. In due course, when the system reaches a point at which it is again no longer meeting the needs of the organisation, it may be discarded and if a new system is needed to replace it the SDLC will be restarted.

Note. Some organisations have systems that have been in use for many years and there may be a reluctance to completely replace these systems with new ones. These are often referred to as "legacy" systems. Often, legacy systems have poorly structured software and limited documentation. In order to continue to use these systems they can sometimes be reverse-engineered by software tools which analyse and restructure the code. This approach aims to provide a better understanding of the software and so aids with its integration with newer systems.

Advantages and disadvantages of approaches / methods

Structured Analysis

Advantages:

Long established

Easier project planning and monitoring

Requirements defined and agreed early in the SDLC

Software designed based on a full understanding of all requirements.

Disadvantages:

Possibility design requirements not what the customer expects

If problems encountered with the system, cost to correct can be high as the client may only see the system when it is complete.

Object-Oriented Analysis

Advantages:

Closer representation of real world objects

Aids development using O-O software languages

O-O software easier to maintain and with greater re-usability potential.

Disadvantages:

Can be difficult to identify all classes and objects

Not ideal for relational database designs

Models not as easy to understand for none technical users.

Agile

Advantages:

Regular system user feedback

Easier to adapt to changing situations

Quicker development.

Disadvantages:

Limited planning

Potential for scope creep – adding features and going over time and cost plans.

Lack of emphasis on design and documentation.

1.3   SUMMARY

Information systems play an important role in supporting an organisation's aims. In order for them to be effective they must be based on a detailed set of requirements which have been identified using appropriate analysis and modelling techniques.

Exercise 1

Which of the following are information systems?

    a) A library catalogue
    b) A motor vehicle
    c) A student name
    d) Ebay.co.uk

Can you think of any other information systems?

Exercise 1 feedback

    a) Yes
    b) No – but it could be part of an information system
    c) No – it is a data value
    d) Yes

Other information systems: Business accounts, stock control, Amazon.com.

Exercise 2

List some possible sub-systems for a university information system.

Exercise 2 feedback

    a) Student admissions
    b) Student details (personal and academic)
    c) Course details
    d) Library loans

Exercise 3

Which of the following is the correct systems development life cycle?

    a) Initiation, design, analysis, testing, implementation
    b) Design, analysis, implementation, maintenance
    c) Initiation, analysis, design, implementation, maintenance

Exercise 3 feedback

    c)

2   SYSTEMS ANALYSIS

On completion of this chapter you should be able to:

- produce a functional decomposition diagram
- model a business situation using data flow diagramming (DFD) techniques
- specify a process using decision trees, decision tables and structured English
- undertake object modelling using the unified modelling language (UML).

2.1   REQUIREMENTS MODELLING

Following the initial investigation and acceptance of the feasibility study, the systems analysis phase of the SDLC will commence. This involves requirements modelling, which includes producing the requirements specification which describes the system to be developed. The specification indicates functional and non-functional requirements and may include a number of models and diagrams that describe how the system will operate. The specification is a formal document which forms an agreement between the developers and the users. In order to help understand the system under investigation and the system that is required, models are produced.

These models include various diagrams, e.g. DFDs or UML diagrams, which are used to communicate with the relevant stakeholders, i.e. users and developers, to ensure that they accurately reflect the system under consideration. The diagrams may be changed a number of times following feedback from stakeholders until all agree that they accurately represent the system. The diagrams are usually produced using a suitable CASE tool, though initial drafts may be drawn by hand. In order to produce the models, the analyst will need to have a good understanding of the business area under consideration and to achieve this will require a number of meetings and observations and possibly other fact – finding activities including business document reviews and user surveys.

Where the new system is to be based on an existing system it is usual to first produce a **physical model** which represents the physical view of the current implementation, in order to gain an understanding of how the system functions. This model can then be analysed and converted to form a **logical model** which allows the analyst to view the actual information required to satisfy the requirements for the new system. Later in the system design phase the logical model can be used to help produce a physical model of the proposed system which will show how the new system will be implemented.

To understand the difference between a physical view and a logical view let us consider a business system that sends invoices to its customers. The physical view represents the actual physical implementation:- the system produces a printed invoice which is then sent in the mail to the customer. However, from the logical viewpoint the system is just communicating invoice details to the customer. By focussing on the logical view the essential information requirements can be established. During the later design stage a decision will be made on how best to implement the above requirement. This may result in the invoice being emailed to the customer or giving them access to the system via the Internet so they can view the invoice details at any time, or both.

2.2   FUNCTIONAL DECOMPOSITION

A major role of the systems analyst is fact-finding, which includes understanding the structure of the organisation. This involves investigating the area of study and breaking it down to identify its business processes, referred to as **"top down decomposition"**. These processes can be shown using a **functional decomposition diagram**. This aids understanding and provides a means of communication with others whilst helping to establish the information requirements. A basic example showing some higher level business functions and some of their processes is shown below. Note that a complete diagram would have more sub-levels showing the business processes undertaken for each functional area:
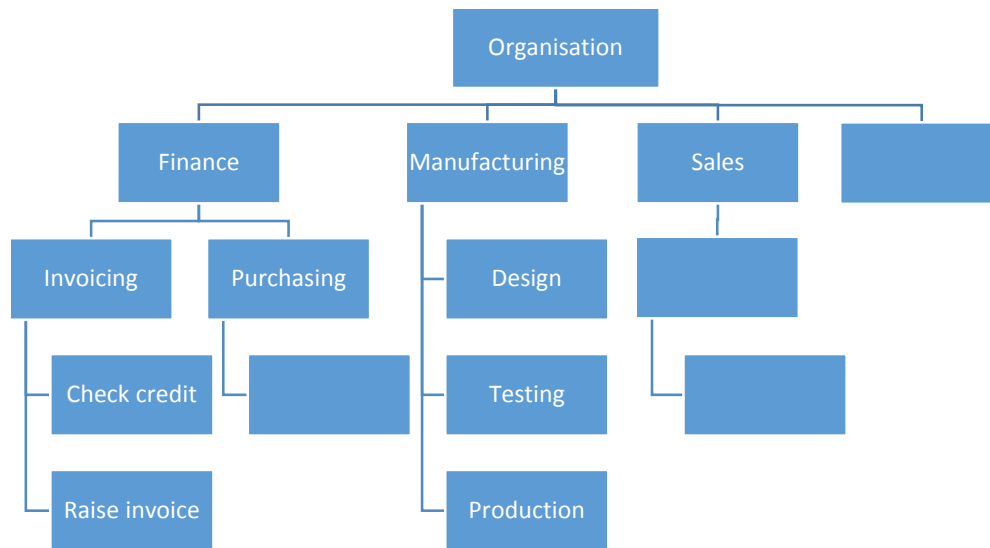
Fig 2.1 Functional decomposition diagram

2.3   IDENTIFYING FUNCTIONS AND PROCESSES

The first step is to break down the business into its main areas of activity or functions. These may be concerned with managing business resources such as Finance, Stock or Human. Alternatively, they may be based on the life cycle of a product or service e.g. Marketing, Manufacturing, Distribution. The major functional areas or activities will depend on the nature of the organisation under investigation.

Business processes usually relate to a specific business entity which is used within a business function. Here are some examples of business processes:-

-   Check Credit
-   Raise Invoice
-   Receive Payment

> **Note the structure of the business processes; they should consist of a verb and a singular object e.g. Pay Employee.**

To identify business processes, consider the life cycle of the business entities e.g. for a hotel you might identify:

-   Receive Booking
-   Check in Guest
-   Allocate Room
-   Check out Guest

The functional decomposition processes will be used within the dataflow diagrams (DFDs).

Exercise 1

What would you suggest are the major activities of :-

a) A library
b) A university
c) A hospital

Exercise 1 feedback

a) Book acquisition, loans and returns, reservations, stock management
b) Marketing, recruitment, enrolment, teaching, examinations, resources
c) Admissions, staffing, patient care, maintenance

Exercise 2

Suggest some business processes for a bicycle manufacturer to be added to those identified below:

-
- Design Bicycle
-
- Produce Bicycle
-
-
- Sell Bicycle
-

Exercise 2 feedback

- **Research Market**
- Design Bicycle
- **Build Prototype**
- Produce Bicycle
- **Advertise Bicycle**
- **Distribute Bicycle**
- Sell Bicycle
- **Bicycle Aftersales Support**

Exercise 3

For the business process **Recruit Employee** decompose the process by completing the following:

    a) Advertise _____?
    b) _____? Candidate
    c) Appoint _____?

Exercise 3 feedback

    a) Advertise Vacancy
    b) Interview Candidate
    c) Appoint Candidate

2.4   DATAFLOW DIAGRAM NOTATION

As mentioned in section 2.1 above, in order to ascertain the requirements for a new system it is important to be able to understand how the current system operates. To achieve this a physical model can be produced using dataflow diagramming (DFD) techniques. The DFD allows you to show business processes and how data flows between them. There are a number of DFD notations in use, e.g. Gane and Sarson, Yourdon De Marco and SSADM®. Although each notation uses slightly different symbols the diagrams are composed and interpreted in similar ways so you should be able to interpret a diagram based on any notation without difficulties after following this text. This text uses the SSADM notation and the diagrams presented have been produced using the QSEE Superlite version 1.1.2 case tool, which is available to download from http://www.leedsbeckett.ac.uk/qsee/

> SSADM® – Structured Systems Analysis and Design Method was a methodology developed for use in the analysis and design of information systems and was widely used for UK government projects from the 1980s onwards.

Depending on the size and complexity of the system being modelled a number of interrelated DFDs showing different levels of detail will be produced in order to show all the information in a manageable form. This involves applying a top-down decomposition approach which identifies the higher level business processes, and expanding each of these to show the lower level processes on separate DFDs, continuing until you have a set of **primitive processes** which cannot be broken down any further.

The DFD uses symbols to show the four key business process elements: **processes**, **data flows**, **data stores** and **external entities**.

A **process** is an activity of interest within the system that involves transforming data and producing an output. It is defined using a verb and an object, e.g. Create Order. In effect, a process represents the business rules that are applied to process data in order to achieve the required output.

The **process** symbol will be one of the following, depending on the chosen notation:
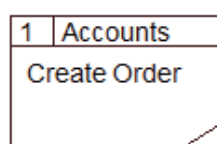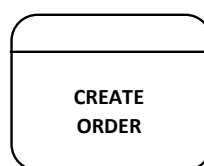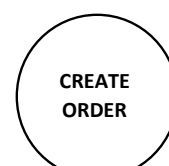
SSADM   Gane and Sarson   Yourdon

Fig 2.2 Process symbols

Each process is given a process name – e.g. Create Order – and an identifier, e.g. 1, 2, 3 etc. In SSADM the process also has a location which indicates the unit or person responsible for carrying out the process e.g. Accounts Department.

A **dataflow** is a single item of data or a logical group of data items that is transferred when a process takes place. This is shown as an arrowed line (in all three notations) showing the direction the data travels, with a suitable label describing the data items. **Note**. Double arrowed lines are permitted but only on higher level DFDs to show that the data items may be transferred in both directions between processes.

ORDER
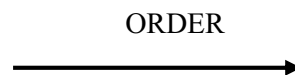
Fig 2.3 Dataflow symbol

A **data store** is a set of data items stored for use by one or more processes over time.

SSADM   Gane and Sarson   Yourdon
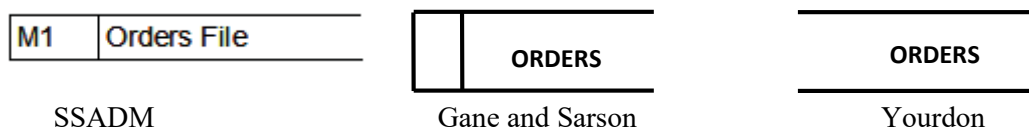
**Fig 2.4** Data store symbols

> **Note.** On a current system physical DFD, data stores are labelled with an "M", "D" or "T". "M" is used to represent a non-computerised store, e.g. a paper file. "T" represents a transient store, i.e. one where data resides temporarily just for use between two processes. "D" represents a data store that is computerised. Each data store label includes a unique identifying number e.g. M1, D1, T1.

An **external entity** is an organisation, person or a system that is outside of the system under consideration but which interacts with it, e.g. a **customer** from another organisation who places an order which is processed by the system under consideration.

SSADM   Gane and Sarson   Yourdon

Fig 2.5 External entity symbol

Note. SSADM external entity identifiers use a lower-case letter e.g. a, b, c.

To aid diagram readability it is permissible to use duplicate data stores and external entities, these duplicate symbols are shown as follows:-

	M1	Orders file

a
Customer

Fig 2.6 Duplicate symbols

Diagram labelling reminders:

- Remember to use a verb and a singular object for processes
- Use clear, meaningful names

Note Gane and Sarson and Yourdon notations use upper case labels, though it is not uncommon to see lower and mixed case labels used on diagrams. However, you should be consistent with your labelling, i.e. use **either** upper case **or** mixed case but not both.

2.5   DRAWING A PHYSICAL DFD

The highest level DFD is called the **context diagram or Level 0 diagram**. This shows the area of activity being investigated and uses only one process symbol, which represents the area (system) and all the external entities which interact with it. The next level down is referred to as **Level 1**. This shows the main processes and their data stores. If any of these processes are further decomposed, their sub-processes will appear on a **Level 2** diagram and so on. Analysts often start with the Level 1 DFD, as it is not usually possible to identify all the external entities and their data flows which are needed for the context diagram until the system has been analysed in some detail. The context diagram and levelling process are explained in more detail later.

Let us look at how these symbols can be used to produce a Level 1 physical DFD to represent the following business process:

"A university loans equipment (e.g. video cameras) to students, on submission of a booking form containing details of what is required. The university technician checks the item's availability against an inventory list before issuing the item, with a loan receipt, to the student."

This is modelled as follows:-
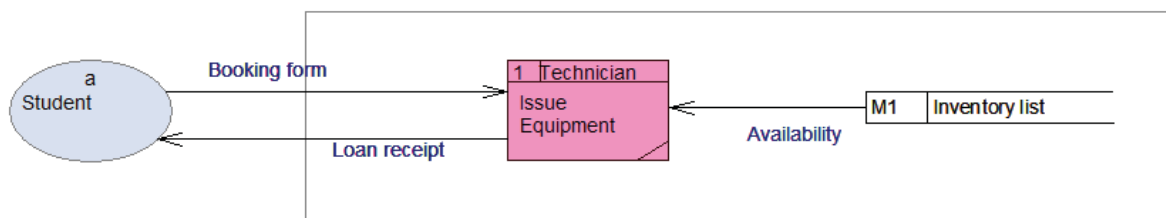
Fig 2.7 Equipment loans physical DFD

This DFD is interpreted as the student's booking form enters the system as an input to the "Issue Equipment" process, which is carried out by the technician. This process involves reading the booking form data (for the item required) and then checking the inventory list which is obtained from the data store (an input to the process) in order to check the requested item's availability. Assuming the item is available for loan, the item and a loan receipt are issued to the student (an output). The actual item loaned is not shown on the dataflow to the student as the DFD is only concerned with showing the data involved, i.e. the loan receipt, as this indicates an item has been issued.

Note. The box enclosing the process and data store shows the boundary of the higher level process (or system) being modelled, although the external entity "student" is beyond the boundary it interacts with the process directly.

Now let us look at a different Level 1 physical DFD scenario which shows how processes are linked:-

A vehicle rental company hires out motor vehicles to clients. The company receives telephone requests from customers to hire a specific car type for a period of time. When a request is received by the company, the Rentals Manager checks to see if a suitable vehicle will be available before issuing a rental confirmation; this involves checking the vehicle file. If a vehicle is available, a rental form is completed and stored in the rentals file.

The Garage Manager uses a copy of the rental form to prepare the vehicle for the client, a copy of which is then given as a receipt to the client when they collect the car. When a vehicle is returned, the garage manager checks the vehicle and signs the copy rental form, before amending the vehicle file to show the vehicle return.

This is modelled as follows:-
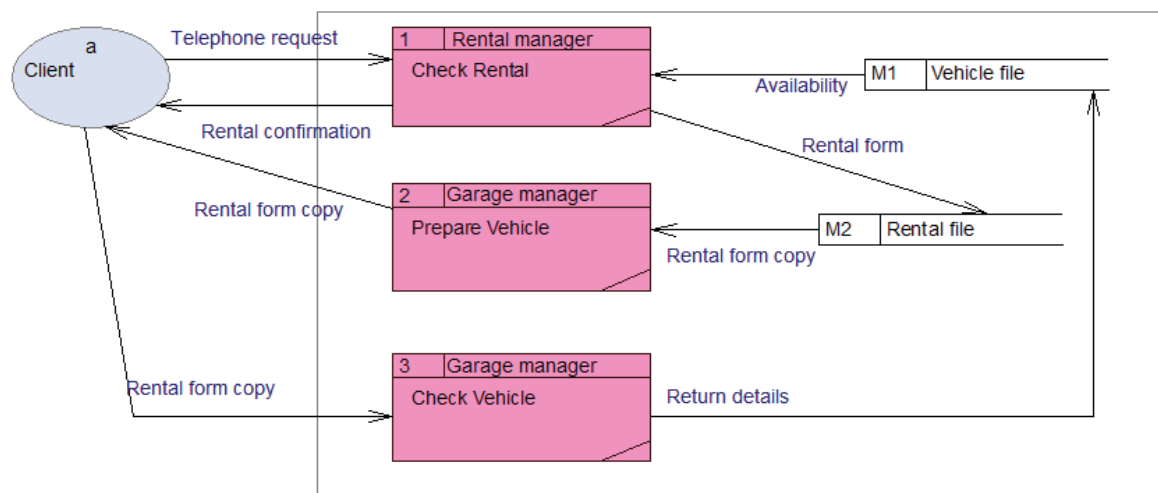
Fig 2.8 Vehicle rentals physical Level 1 DFD

Note The physical DFD is so called because it shows the physical nature of the data objects, e.g. Order form, Sales folder etc.

2.6   DFD ERRORS

When drawing DFDs you must ensure that all external entities, processes, data stores and data flows are clearly labelled and that dataflow lines do not overlap – use duplicate symbols if necessary to avoid this.

Take care to avoid incorrect data flows when producing your DFD, as shown by the following examples.

1.
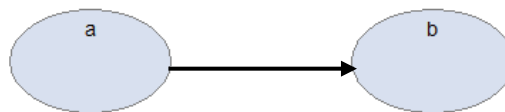
Direct interaction between external entities is not permitted. This must be via a process if required.

2.
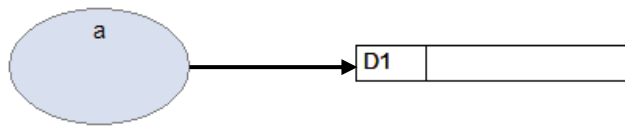
Data from an external entity cannot be placed directly into a data store – it must be transferred via a process.

3.
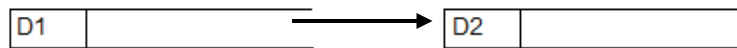
Data cannot be transferred directly from one data store directly to another – it must be transferred via a process.

4.
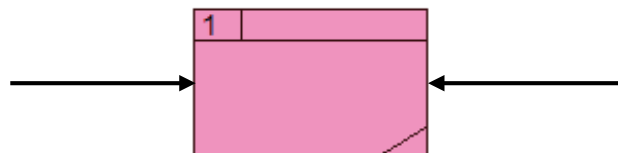
A Black hole – data must be transformed by a process, there must be an output to a data store, another process or to an external entity.

5.

Divine intervention – in order for data to be output from a process, some data must be input.

6. A Grey hole is one where an input would not be able to produce the required output e.g. to calculate an employee's pay, inputting a car registration number would not be relevant.

7. A trivial process is one where no data transformation is taking place within the process e.g.:-
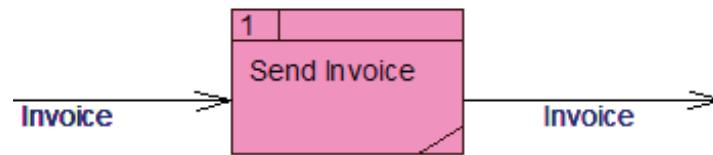
Every process must have at least one relevant input data flow and one output data flow in order to "transform" the data.

Exercise 4

Complete the following Physical DFD example which represents part of a cycling club enrolment process.

"The secretary at the cycling club enrols applicants as members as soon as he receives their signed application form. He updates the members' file with their form then issues a membership card. This completes the membership process."
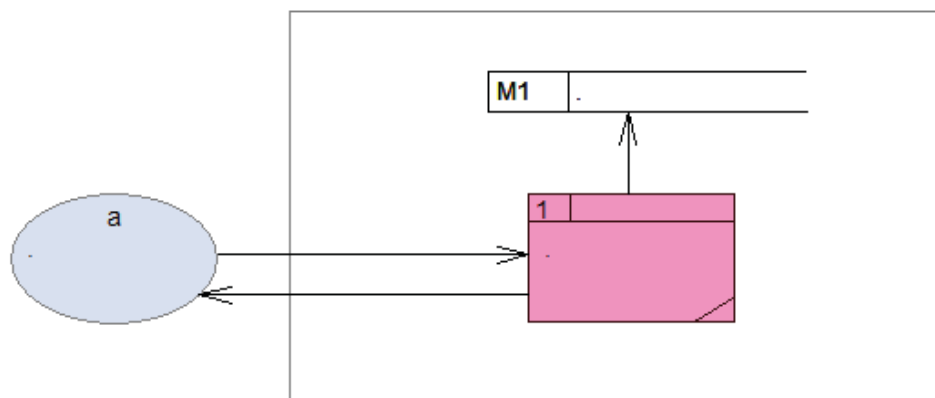
Exercise 4 feedback
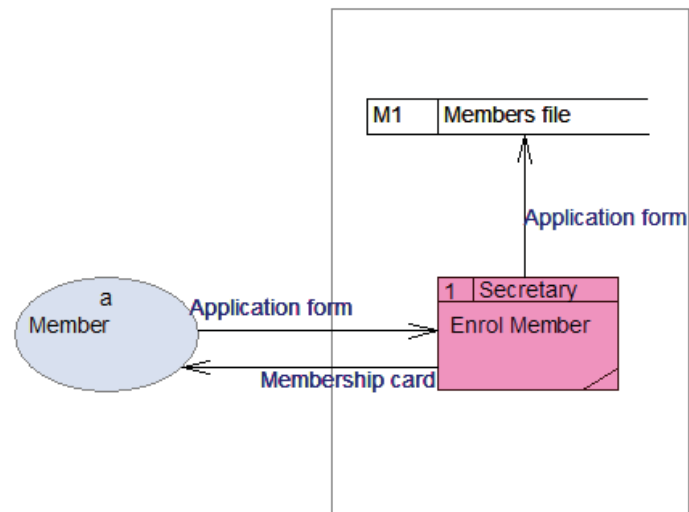
Note The external entity could alternatively have been labelled as "Applicant". Although at the outset of the process the external entity is an applicant, after being accepted they would become a member and any further interactions with the system would be as a member. The "application form" data flow to the members' file could alternatively be labelled "member details" if these are stored instead of the actual application form.

Download free eBooks at bookboon.com

2.7   DRAWING A LOGICAL DFD

The DFDs shown previously have been compiled in order to give a structured view of an area of activity from a physical perspective, e.g. Booking form and Loan receipt. As an analyst you need to be able to detach the physical aspects of the existing system in order to obtain a logical view of the system to be developed. It is essential to be able to identify the logical requirements for the new system and not be constrained by the existing system's implementation.

Let us now look at how you can "logicalise" a physical DFD to produce a logical DFD.

Use the following steps:

1. **Data flows** – examine and replace any mention of a physical representation, e.g. documents such as forms, with their data items.
2. **Data stores** – rename any which refer to physical documents or records. Consider using business data entity names for the stores, e.g. 'Customers' rather than 'customer file'. Also ensure that each logical data store is uniquely identified with a "D" e.g. D1, D2 etc.
3. **Processes** – Check each process and remove any reference to people or places (the location) and remember the processes should not be trivial – they must represent some data transformation.

Here is a simple example to start with – the cycling club enrolment, using the physical model shown in exercise 4 above:-
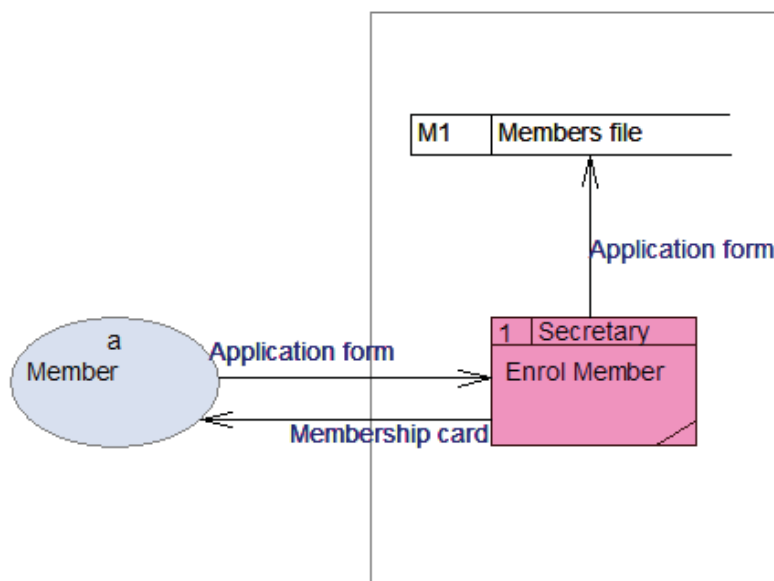
Fig 2.9 Enrol member level 1 DFD

Following the three steps:

1. First look at the data flows:-
   Application form – this is a physical document name. The data needed by the process is actually the applicant member's information e.g. name, address etc. This can be referred to as "member details".

   Membership card – this is also a physical item. You could rename this as "membership confirmation". In the new system this may take a different form such as an electronic identification that may be used on a mobile phone or other device.

2. Data stores:-
   Members' file: The business entity here is "Member" but as you wish to store all the required information about all the members in the club here then the name "Members" is appropriate. As all logical data stores are identified with a "D," its identifier will be "D1".

3. Processes:-
   Enrol Member: This is acceptable (verb and object), however no location should be specified so "Secretary" would be removed. The reason for this is that in the new system the process may be carried out automatically by the system.

The new logical DFD is as follows:-
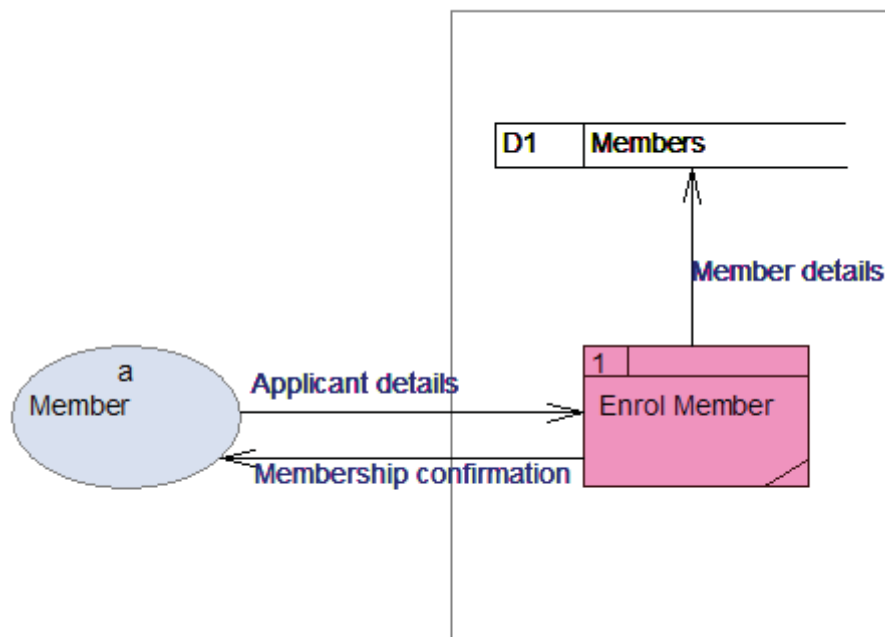
Fig 2.10 Enrol member logical level 1 DFD

Note that the data flow from the process to the data store has been renamed "Member details" rather than "Applicant details" as this is a more appropriate description of the data at this point in the process.

> The "logical" model describes the processes and information that are relevant to the business or, more specifically, what is needed rather than its form. It forms the basis of further analysis and design activities.

Exercise 5

Convert the physical level 1 DFD for the vehicle rentals company shown in Fig 2.8 to a logical level 1 DFD.

Exercise 5 feedback

Logical level 1 DFD
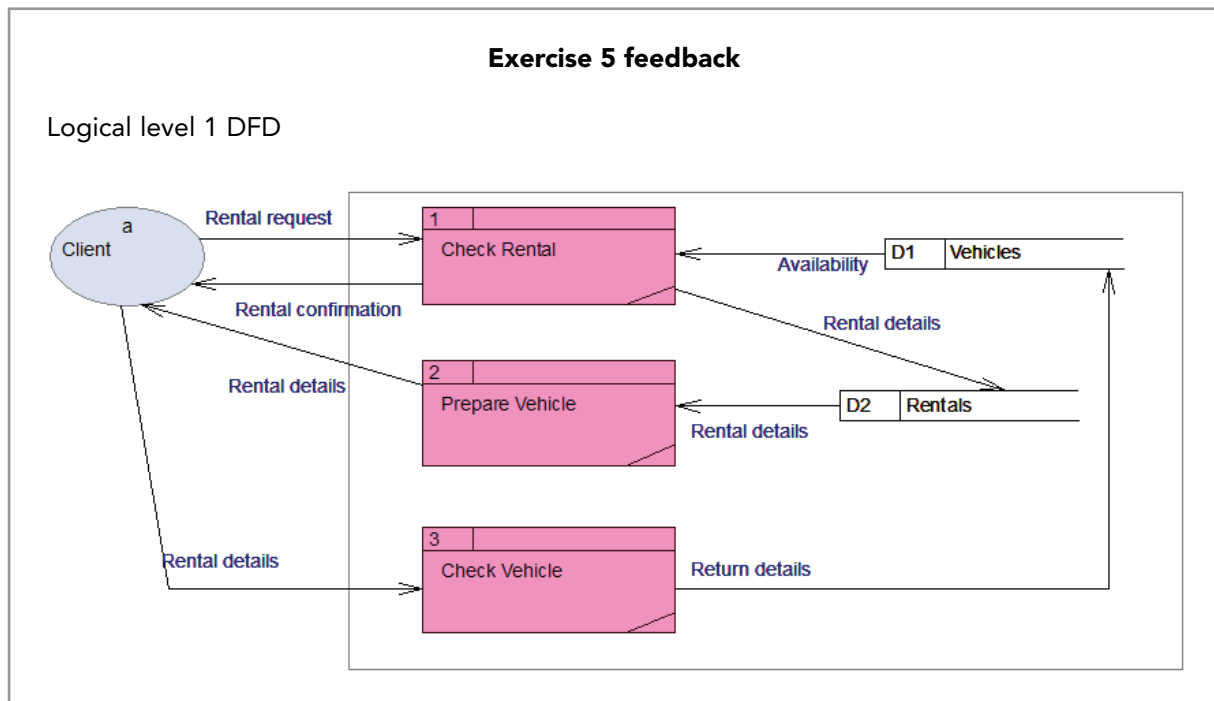
2.8   LEVELLED DATA FLOW DIAGRAMS

As has already been mentioned, using the DFD structured diagramming technique allows for a top-down approach to modelling so that a DFD can be decomposed (expanded) into lower levels, each of which shows more detail for each process by separating these into their sub-processes.

The highest level diagram – Level 0 – is also called the context diagram as it shows the boundary for the whole system as one process, surrounded by the external entities (the context) and their data flows that represent the interaction with the system.

The Level 1 DFD shows the main processes and data stores for the area under investigation.

A Level 2 DFD is normally produced for each process shown at level 1, though it is not always necessary to break every process down to the next level.

Decomposition stops when processes on a diagram are regarded as primitive or elementary i.e. they can be specified simply using another method such as "Structured English". (This is discussed later.)

Processes are numbered on lower levels using the following convention.

If a Level 1 DFD had a process numbered 1 which consisted of 3 sub processes, these sub processes would be shown on a Level 2 DFD numbered 1.1, 1.2, 1.3.

If a second Level 1 process with an identifier of 2 had 2 sub processes, these would be labelled as 2.1 and 2.2 on another Level 2 DFD.

The process at a given level sets the boundary for the DFD at the next lower level. It is important that the data flows shown on the lower level DFD match those shown on the higher level. This is referred to as "**balancing**" the model. CASE tools help to check that this "balancing" has taken place as they usually transfer the data flows (and data stores) automatically down to the next level when the subordinate diagram is created.

If a data store is only used by one process (i.e. it is "private" to that process) it would only appear at the lower level.

External entities are shown at the lower levels if data flows are connected directly to processes at the lower level.

Let us consider the car rental Level 1 DFD, which has three processes. The "Check rental process" actually involves three sub processes: check vehicle availability, create rental booking and inform the client. These sub processes can be shown on a Level 2 diagram as follows:-
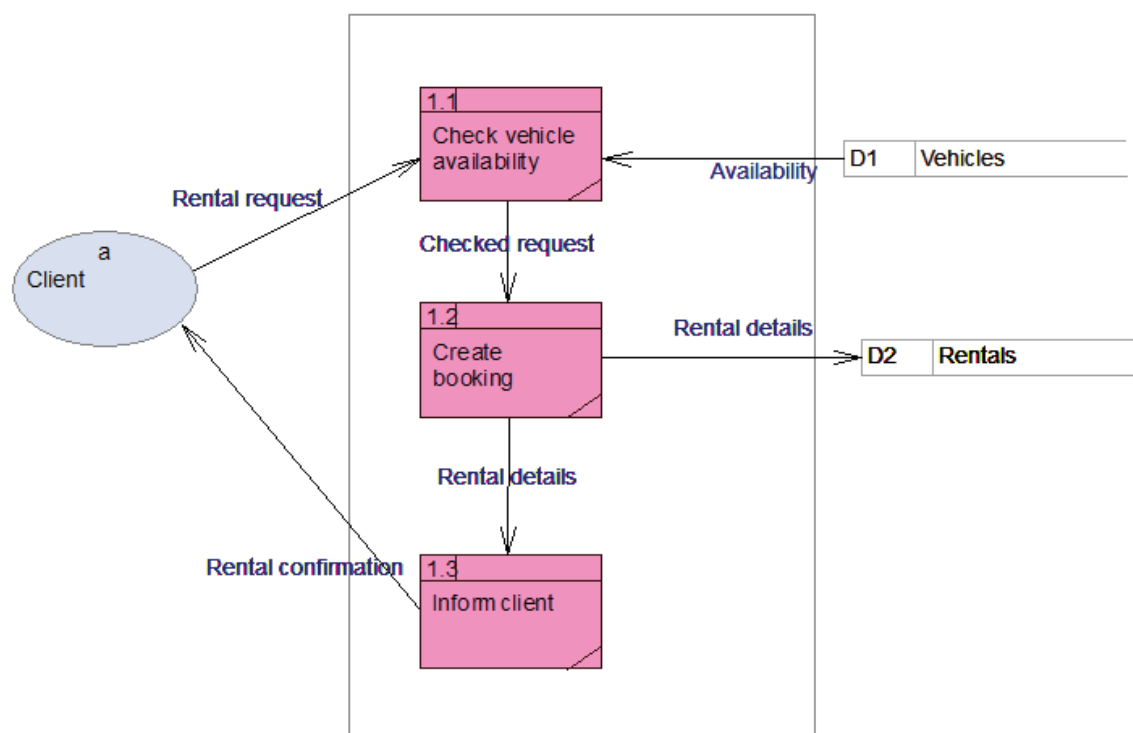
Fig 2.11 Check rental level 2 logical DFD

The box around the three processes indicates the boundary of the level 1 process which is represented by the Level 2 diagram.

Note the consistency between the Level 1 and Level 2 diagrams for the "Check rental" process.

1. In and out data flows are identical on the Level 2 diagram with the process 1 of the Level 1 diagram.
2. The two data stores involved with the Level 1 process "Check rental" match at Level 2 as shown outside the Level 2 boundary (the large box)
3. The external entity "Client" appears at Level 2 as it is directly associated with the Level 1 process "Check Rental" on the Level 1 diagram.

On the Level 2 diagram there are new data flows that link the processes together.

Note. If the "Inform client" process also involved storing a copy of the confirmation sent to the client, this would be represented as a "private" data store called "Confirmations" and linked with an output data flow from this process to the Client entity. It is referred to as "private" because it would only be visible at this level if not used by any other processes at a higher level.

2.9   THE CONTEXT (LEVEL 0) DIAGRAM

The first diagram in the hierarchy is the Context diagram, which is also referred to as the Level 0 DFD. This diagram shows the entire area of activity under consideration (the system) and is often referred to as the scope of the analysis or investigation.

Logically, the context diagram is drawn first so that the extent of the area of investigation can be identified and the external entities and data flows shown. However, in practice it is usually easier to start with the Level 1 diagram and then transfer data flows from this and link them into the single process box on the context diagram.

Using the Level 1 logical DFD for the vehicle rental company would produce the following context diagram:-
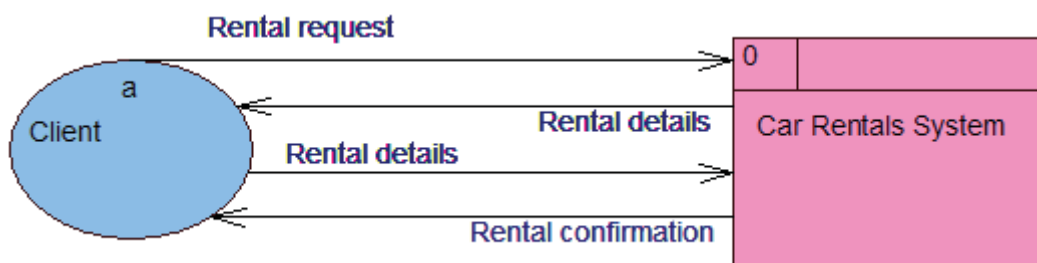
Fig 2.12 Car rentals context diagram

Note the following:-

- Only one process symbol appears on the context diagram labelled with the system name
- The process has the identifier number 'zero'
- No data stores are shown on the context diagram
- There are four data flows, which match those on the Level 1 DFD for compatibility

As some context diagrams have many data flows, amalgamation is permitted to aid readability, e.g. the two "Rental details" data flows may be shown below as one but with arrow tips at both ends of the flow (note this is only permitted on a context diagram).

Rental details
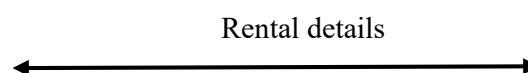
Fig 2.13

> **Important**
>
> Always ensure compatibility with the higher level data flow diagram –
>
> - Show the boundary to represent the higher level process
> - Check there are the same number of data flows in and out on each level
> - Check the same names are used for data flows
> - Check external entities are compatible

Note If using the QSEE CASE tool you will need to enter the context diagram first, although it is not always possible to identify all the external entities or data flows at this stage. This is not a problem as the CASE tool will ensure that any external entities or data flows that are added to the lower level diagrams, e.g. Level 1 DFD, are automatically added to the higher level diagram(s) to ensure the diagrams remain "balanced".

Exercise 6

Produce a context diagram for the cycling club as shown in the Level 1 logical DFD Fig 2.10

Exercise 6 feedback
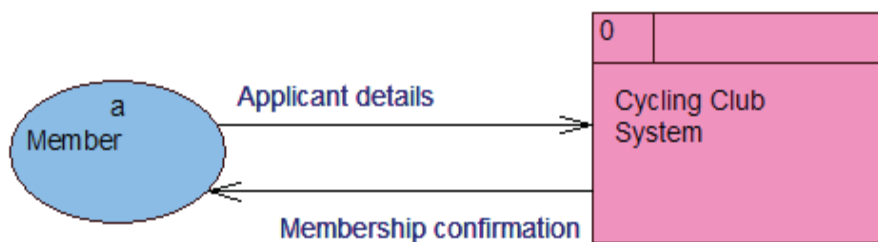
2.10 THE DATA DICTIONARY

As the analyst builds the logical model, additional information needs to be captured and stored for future reference. This collection of information is referred to as the **data dictionary** or **data repository** and is normally stored within a CASE tool, as this allows for easy reporting and cross checking to see where data items are used. The information collected includes the data values from the data flows, data stores and the processes. These data values include **data records** or **data structures**, each of which comprises individual **data items** or elements.

An example data set (record) for a Client in the vehicle hire system would include the data items

- client name
- client address
- client home/office telephone number
- client mobile phone number
- client driving licence id.

Data items are further defined to include a **data type**, e.g. numeric or textual, and a **size**, e.g. maximum number of characters. Additional related information that is often stored includes expected data volumes, i.e. how many customer records need to be stored, or how many transactions will be processed in a given period of time, e.g. how many orders will be processed per day. For each data item, acceptable values or ranges will be noted along with security details relating to user access rights. All of this information is referred to as **meta-data** and is vital for use in later stages of the SDLC.

2.11 PROCESS SPECIFICATION

Each elementary process from the lowest level DFD needs to be specified in a suitable form that clearly describes the processing necessary to accomplish its purpose. This information forms the basis for software specifications which are used in the system design phase. There are three main modelling techniques available for specifying this process logic: **Decision Trees, Decision Tables** and **Structured English**.

2.12 DECISION TREES

A decision tree is used to present a graphical representation of all possible conditions and their actions, as in the following process description which has been identified by a systems analyst during an interview with the rental manager for a vehicle rental company:

"There are two types of client who rent vehicles, business clients and private clients. Private clients have to pay for the vehicle rental at the time of the booking, however business clients are given credit terms. If they have been a client for three or more years, they are allowed credit up to £10,000; those who have been clients for less than this are allowed £5000 credit."

This text shows that a number of decisions are made, with set actions being taken depending on the outcome. Each business situation is identified, in this case whether the hirer is a business or private client. Each of these may have further branches which represent further sub-division points. At the ends of the final branches, the action to be taken is specified as shown here:
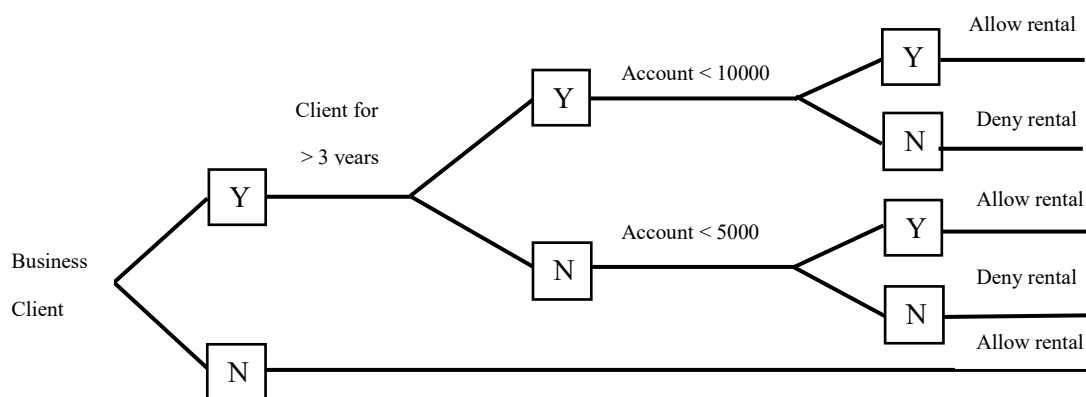
Fig 2.14 Decision tree

2.13 DECISION TABLES

A decision table uses a tabular approach to show every condition and its outcome. These can be set up easily using a spreadsheet program. They are often used by analysts to show how to process a number of complicated conditions in a form that is easy for others to understand. In particular, software developers can use them to help develop the code during the system design phase of the SDLC.

The table consists of four segments as shown below:-
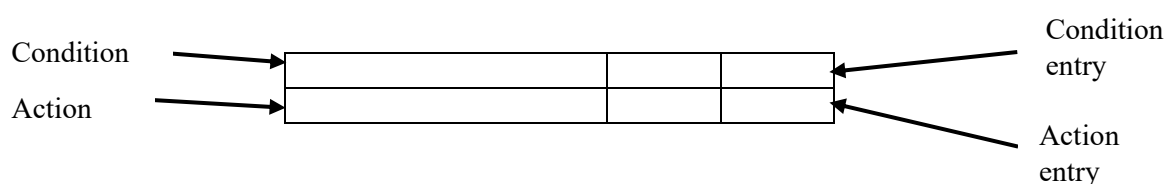
Fig 2.15 Decision table

The condition segment lists the conditions or "questions". The condition outcomes are either "True" or "False" and are represented by a "Y" or "N"; a dash "-" is used when the outcome is immaterial. Each condition entry is referred to as a **rule** and these are usually numbered.

The condition entry segment is used to record the responses to the condition questions as Y", "N" or "-".

An "X" is entered into the action entry alongside the action to be taken under the condition outcome specified above it. The action entry is left blank if no action is to be taken.

To create a decision table, it is necessary to list all the conditions and note the alternative answers.

Then calculate the total number of alternatives of the conditions, e.g. for three conditions, each has two outcomes ("Y" or "N") so this gives $2 \times 2 \times 2 = 8$.

List the actions in the action segment and place an "X" in the action entry boxes relating to the appropriate rules.

Let us consider an example process:-

A bank's decision on whether to dispense money to a customer from a cash dispenser involves checking that the amount to be withdrawn does not exceed the customer's account balance.

	1	2	3	4
Balance in credit	Y	Y	N	N
Withdrawal amount > balance	Y	N	Y	N
Dispense money		X		
Reject request	X		X	X

Fig 2.16 Bank decision table

It is sometimes possible to consolidate the rules when the actions specified for two sets of rules are identical and the condition entries differ for only one condition. In this example the two rules can be amalgamated into one. This is accomplished by replacing the "Y" of the condition with a "-" and deleting the rule which has an "N". Revisiting the previous bank example allows for simplification as follows:

	1	2	3
Balance in credit	Y	Y	N
Withdrawal amount > balance	Y	N	——
Dispense money		X	
Reject request	X		X

Fig 2.17 Consolidated bank decision table

If the balance is not in credit, then the condition "withdrawal amount < balance" is irrelevant as the action will be the same for both conditions, thereby reducing the number of condition entries in the table.

Exercise 7

Convert the vehicle rental company decision tree from Fig 2.14 into a decision table.

Exercise 7 feedback

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Business client	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Client for 3 or more years	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
Account within £10,000 credit limit	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Account within £5,000 credit limit	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Allow rental	X	X			X				X	X	X	X	X	X	X	X
Deny rental			X	X		X	X	X								

The above decision table shows all 16 possible combination rules but when the rules are consolidated the result shows that only 5 rules are needed:-

	1	2	3	4	5
Business client	Y	Y	Y	Y	N
Client for 3 or more years	Y	Y	N	N	–
Account within £10,000 credit limit	Y	N	–	–	–
Account within £5,000 credit limit	–	–	Y	N	–
Allow rental	X		X		X
Deny rental		X		X	

Remember the dash "-" indicates that the condition entry value is not relevant.

2.14 STRUCTURED ENGLISH

The final technique that can be used for describing a primitive process is Structured English. Whilst decision trees and decision tables show the different branch logic, Structured English can be used to describe all processing steps. It adopts a modular design approach and although it looks similar to **pseudocode** which is used by software developers to help specify program code, it is not the same, as it is used to identify the main process logic rather than specific code. Structured English utilises the three logical control structures: **sequence, selection and iteration**. These structures are combined in order to define the processing logic. Structured English aims to provide a clear explanation of the processing involved, which aids the software developer when designing the code during the systems development phase of the SDLC.

Structured English uses three logical control structures:

Sequence A sequence is where a number of statements are executed one after the other. This can be shown diagrammatically as follows:-
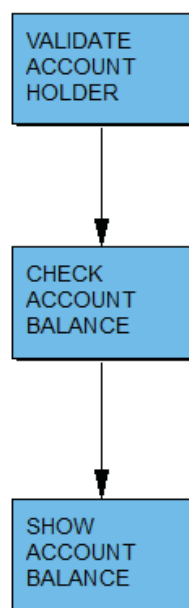
Fig 2.18 A sequence structure

Note. Any step of a sequence may consist of other sub-processes which contain logical structures.

Selection. A selection checks for a condition and, depending on the outcome, performs one of two steps as follows:-
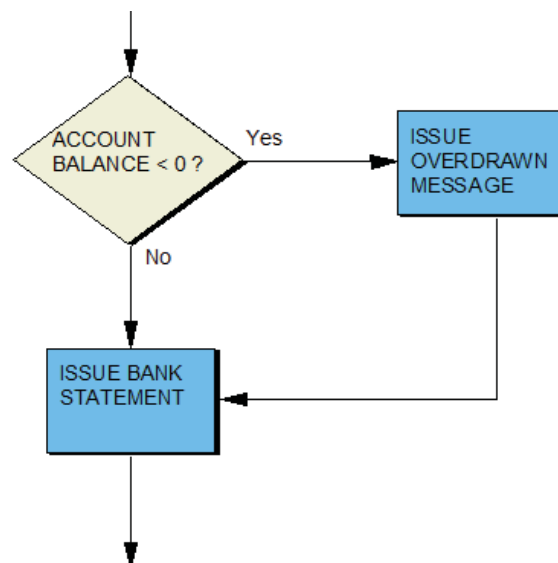
Fig 2.19 A Selection structure

Iteration (Looping) A process step or steps repeated until a condition is met:-
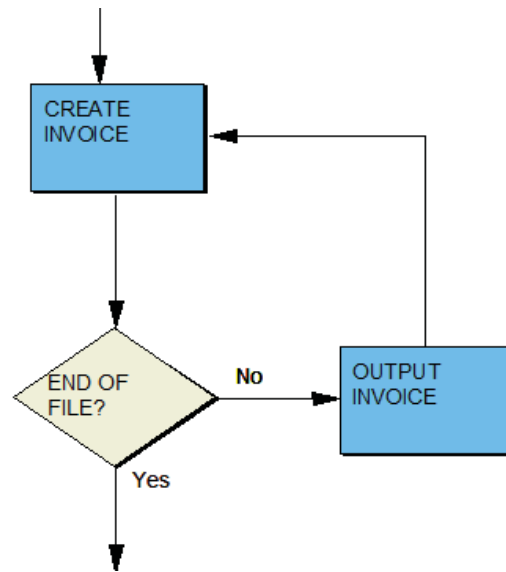
Fig 2.20 An Iteration structure

Below is a Structured English example. Note that a number of key words are used along with references to the data items (they are defined in the data dictionary) being processed.

Typical key words which are used include the following:- READ, CREATE, UPDATE, WRITE, DELETE, OUTPUT, IF, ELSE, ENDIF, REPEAT, UNTIL, WHILE, THEN, DO WHILE, END WHILE, EQUAL, LT (LESS THAN), LE (LESS THAN OR EQUAL TO), GT (GREATER THAN), GE (GREATER THAN OR EQUAL TO), AND, OR, NOT, OPEN, CLOSE, FILE, EXIT.

```
CREATE INVOICE

*This process produces an invoice for every completed order; incomplete orders are reported

REPEAT UNTIL (end of file)

READ order file

        IF (order complete)

                OUTPUT invoice

        ELSE

                Add order no. to incomplete order report

ENDIF

ENDREPEAT

EXIT CREATE INVOICE
```

Fig 2.21 Structured English process description

> **Note** When using structured English only use the three constructs. Indent statements for clarity. Place one sequence statement per line. Always use UPPER case for keywords. Underline or highlight words relating to data dictionary entries. Comment lines start with a *. Use ( ) to enclose conditions.

2.15 REQUIREMENTS CATALOGUE

Following identification of the system requirements they will be documented as a set of requirements catalogue entries using a template similar to the one below. These will be used in conjunction with the structured analysis and object oriented model diagrams that have been produced and are usually stored within a CASE tool repository for easy reference.

Requirements Catalogue Entry	
System	
Author	
Requirement Number	
Version Number	
Requirement Name	
Description	
Priority	
Source	
Owner	

Issues and Outstanding Questions	
Comments/Suggested Solutions	
Benefits	
Related Documents	
Related Requirements	
Resolution	

Fig 2.22 Requirements catalogue entry

The template fields are completed as follows:

- System – the proposed system name
- Author – the person who has documented the requirement, usually the analyst
- Requirement Number – the number or ID which uniquely identifies the requirement
- Version Number – used to indicate which version this requirement is as they may be amended
- Requirement Name – a brief name
- Description – a description of the functional (essential) requirement in the user's words
- Non-functional requirements may be included, e.g. transaction response times which may include acceptable ranges
- Priority – e.g. Low, medium, high. Not all requirements are implemented so it will be necessary to agree essential ones
- Source – person or document that identifies the requirement
- Owner – person who will take responsibility for the requirement
- Issues and Outstanding Questions – issues to be considered
- Comments/Suggested solutions – ideas for consideration
- Benefits – need to relate to business priorities and the Priority mentioned above

- Related Documents – refers to other analysis and design documents that have been produced, e.g. models/diagrams
- Related Requirements – refers to other requirements that relate to the same functional area of the system
- Resolution – indicate the outcome of the requirement, i.e. completed, if discarded, state reason.

Finally, the **software requirements specification (SRS)** document is produced which incorporates the above diagrams and documentation. For an example SRS template see (Wiegers, 1999). The SRS is an essential document that is used in the next phase of the SDLC – Systems design.

2.16 SUMMARY

Identifying system requirements is not a simple process and requires observation and questioning of a range of stakeholders to ascertain their requirements, some of which may be contradictory or vary in priority.

Analysts often produce a physical model of an existing system in order to understand its function and then produce a logical model. This is often followed by a logical model of the proposed system, which in turn may lead to the development of a physical model representing the proposed new system.

Process modelling involves producing a number of sets of diagrams. These diagrams include DFDs which enable the analyst to understand a system from a physical and a logical perspective. Complex systems require a hierarchy of DFDs; the lower the level, the more detail is shown. At the lowest level elementary processes are described using decision trees, decision tables and Structured English. As the DFDs are being produced the data requirements are also being identified and these are stored within a data dictionary for reference in later stages of the system's development life cycle.

At the conclusion of the analysis phase the new system requirements have been identified and documented. These are then referred to during the next phase of the SDLC – systems design.

3 OBJECT ORIENTED ANALYSIS

On completion of this chapter you should be able to:

- understand what is meant by Object Orientation (O-O)
- understand key O-O terms
- produce common O-O analysis diagrams
- understand a Business Process Model (BPM) diagram.

Object oriented analysis modelling provides an alternative to structured analysis modelling. It views the system as a set of objects and how they interact. Object Oriented Analysis and Design (OOAD) has become more widespread with the increased use of object-oriented programming languages such as Java and Python.

The **Unified Modelling Language (UML)** (Introduction To OMG's Unified Modeling Language® (UML®), n.d.) includes a number of diagrams (UML2 has 13) for describing a system from an object perspective and is widely used for **object modelling**, so helps improve communication between analysts and users. This chapter aims to introduce you to the most popular UML diagrams in their basic form. A full description of the UML is beyond the scope of this book, however further reading references are provided.

In order to produce an object model, you will need to understand the following O-O terms.

3.1   OBJECTS AND CLASSES

An **object** represents a real world item such as a bicycle or a person, and it contains both **data** and **methods** (operations) for processing the object's data. The object's data is stored as a set of **attributes** and these data values are hidden so as to prevent direct access from outside; this is called **encapsulation**.

Objects have different states during their lifetime and these states are altered by events.

Objects belong to a **class**, which is a group of similar objects. A class consists of a name, a set of attributes and a set of methods and is represented as shown below. For example, a racing cycle belongs to the class named *bicycle*. A class is the specification template for a set of objects and has attributes and methods. Each object (instance) of the class will have a particular set of attribute values and any instance of the class will be able to carry out the class methods in response to the appropriate message.

Class name ⟶ + PERSON

Class attributes ⟶
- Name
- DOB
- Sex
- Address
- Tel No.

Class methods ⟶
+ Change address()
+ Change Tel No()
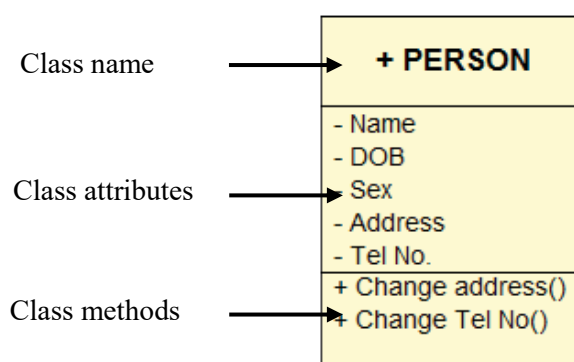
Fig 3.1 shows an object class

Example objects (instances) of the PERSON class:-

Barbara De Silva 21/02/86 Female 7 Av. Atlântica Rio DeJaneiro Brazil +55 21 2323232

Mike Smith
12/06/87 Male 6 Long Row Leeds United Kingdom +44 770454545

Ahmet Aslam 12/03/85 Male At Meydanı No:222 Sultan Ahmet Istanbul Turkey +90 202 527 0239

Fig 3.2 Person Objects

Exercise 1

Identify three business classes for a university system. For each class identify three attributes and two methods (operations).

Exercise 1 feedback

Class	Attributes	Methods
Student	Student name Student address Student D.O.B.	Amend address Enrol on course
Module	Module name Module length Module level	Add module Alter module level
Lecturer	Lecturer name Lecturer office Lecturer telno.	Add lecturer Amend office

Object classes can be arranged into a hierarchy in which a "sub-class" inherits the characteristics of the "super-class" The sub-class has all the attributes and methods that the super-class has. This is called **inheritance**. The sub-class can have additional attributes and methods and can be broken down further into more detailed sub-classes.

To alter data within an object a **message** must be sent to the object asking it to perform an appropriate method. A message "add person" would cause the PERSON class to add a new instance of PERSON with values for its attributes (providing class PERSON has a method to add new person). Likewise, a message "Enrol" sent from LECTURER to STUDENT would enrol the student on a course.
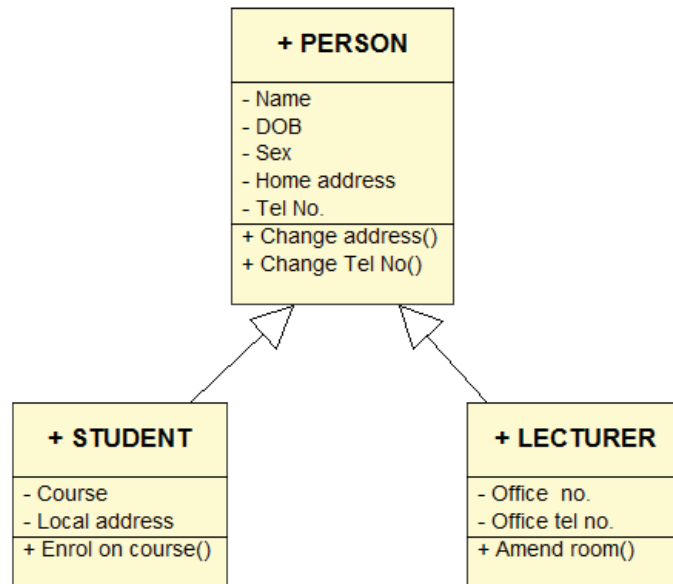
Fig 3.3 Sub-classes and super-classes

In the above example the STUDENT and LECTURER classes are sub-classes of the super-class PERSON and inherit the attributes and methods of the super-class.

A particular message can be sent to different objects and result in different actions being taken. For example, the following model shows a class SYMBOL which has sub-classes SQUARE and CIRCLE. If the message "Draw" is sent to the SQUARE object a square shape will be drawn; however, if the same message is sent to the CIRCLE object a circle will be drawn. This ability to interpret messages differently depending on the object is called **polymorphism**.
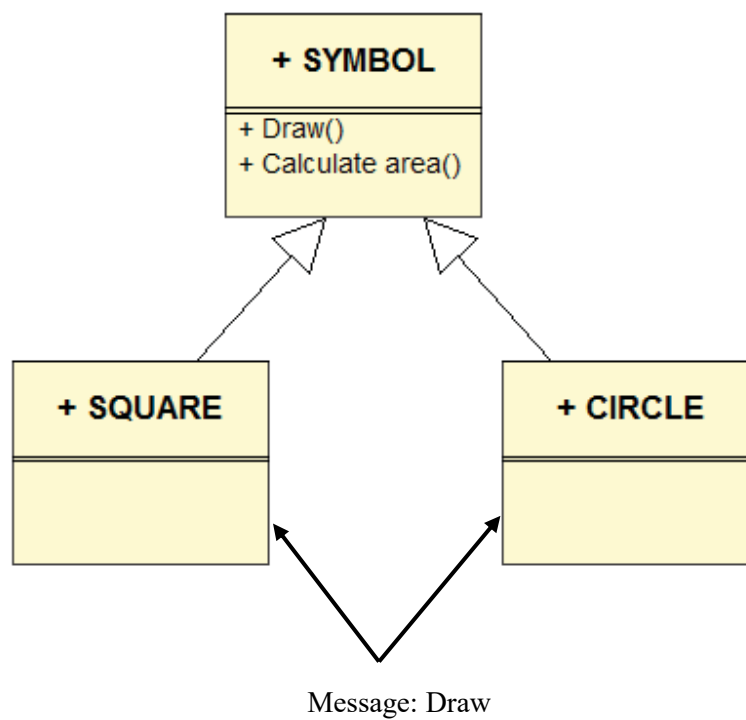
Fig 3.4 Polymorphism

Exercise 2

Identify two sub-classes for the STUDENT class shown above in Fig 3.3. For each of the sub-classes identify two attributes and two methods which would not be in the super-class student.

Exercise 2 feedback

Sub-class	Attributes	Methods
Undergraduate student	Personal tutor Student status	Assign personal tutor Amend status
Postgraduate student	Research group Research supervisor	Add supervisor Amend research group

Full-time and Part-time students could also be considered for sub-classes of STUDENT.

Object relationship diagrams provide an overview of the system objects and how they interact, and can provide a useful overview of the system.

Fig 3.5 Object relationship diagram

3.2   USE CASE MODELLING

The UML includes **Use Case Modelling** which is widely used to show the functionality of a system from a user's perspective and provide a useful way of identifying and documenting the requirements for a system. A **use case** diagram comprises a number of **scenarios**. A scenario describes how a user interacts with the system in certain situations. An external entity called an **actor** (a person or other system) interacts or communicates with a **use case** which carries out a process or function.

To illustrate how a use case diagram is formed, here is an example from the point of view of a hotel receptionist. The scenario for booking a hotel room for a guest would be:

"Ask the guest what type of room they would like and how long they wish to stay. Check the bookings system to see what rooms are available. Confirm a room and dates with the guest. Enter the booking details on the system."

Note how this is written from the point of view of the system user (the receptionist). However, consider what happens if the receptionist is unable to offer a room of the type requested.

Another scenario can be specified to deal with this situation, for example the guest may be offered a different grade of room. The "Book Room" **use case** would consist of the main success scenario (expected to deal with most cases) and an extension scenario to deal with the exception. (**Note** There may be a number of extension scenarios depending on the situation).

Book Room

Main success scenario

1. Ask guest what type of room they require
2. Check bookings system for room availability
3. Confirm room booking with guest
4. Update the bookings system.

Extensions (additional scenarios or exceptions)

1a. Room type not available

      1a1. Identify alternative available room type

      1a2. Return to step 3 in the main success scenario.
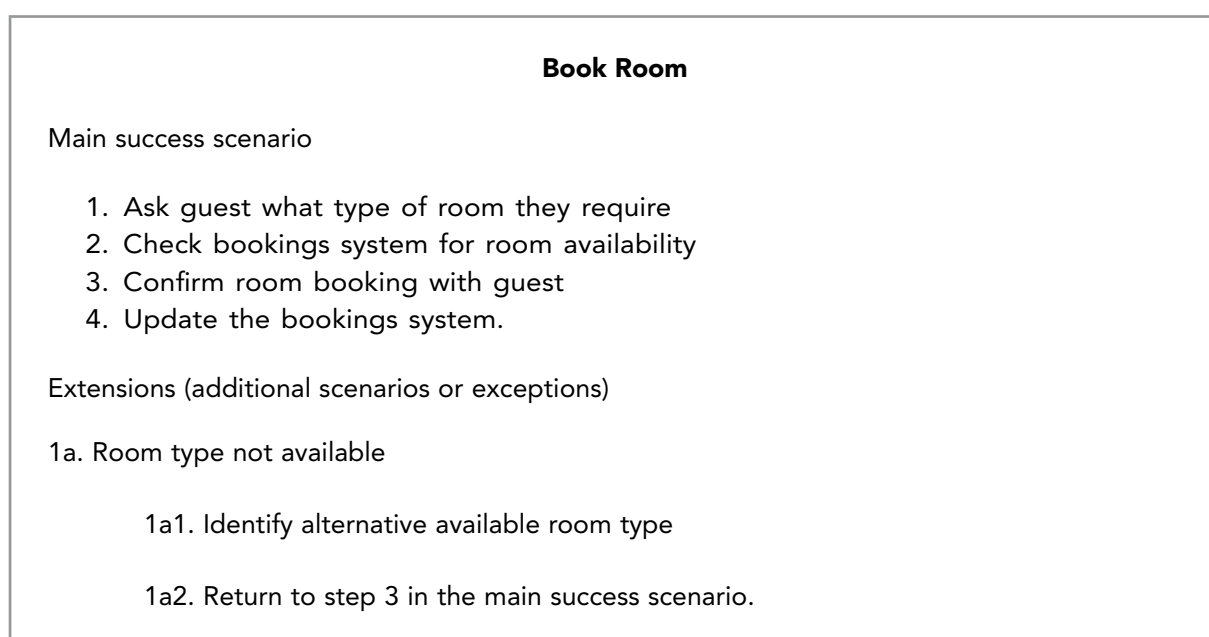
Fig 3.6 A simple use case description

Although a simple description similar to the one above may be suitable for a simple use case or for a small system, for more complex systems, analysts often produce a more comprehensive description which includes additional information. The following template illustrates this for the scenario above.

Use case name	Book room.
Scenario	Book hotel room for guest.
Trigger	Guest requests a room.
Description	When a guest requests a room type the receptionist checks the bookings system to see what rooms are available. If a room of the type requested is available a booking is confirmed.
Actors	Hotel receptionist.
Related use cases	Check room availability.
Stakeholders	Hotel management.
Pre-conditions	Guest requiring a room.
Post-conditions	Room booking confirmed.
Main scenario	1. Ask guest what type of room they require 2. Check bookings system for room availability 3. Confirm room booking with guest 4. Update the bookings system.

Extensions	1a. Room type not available
	     1a1. Identify alternative available room type
	     1a2. Return to step 3 in the main success scenario.

Fig 3.7 Extended use case description

Note. There are many **use case** description templates in use. However, they all include the main information requirements.

A simple **use case** diagram based on the above scenario is shown below.
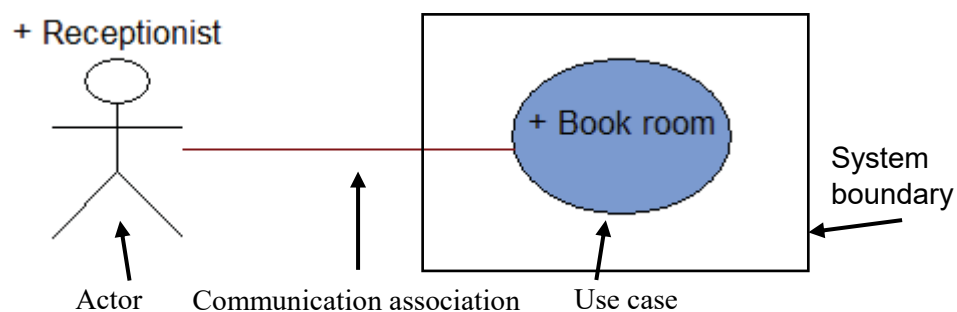
Fig 3.8 A simple use case

Use cases can also interact with other use cases. This is the case when one use case **includes** the functionality of another use case. It is shown by using the **<<include>>** relationship. The following example shows that the use case for checking room availability would be incorporated into the use case for booking a room.
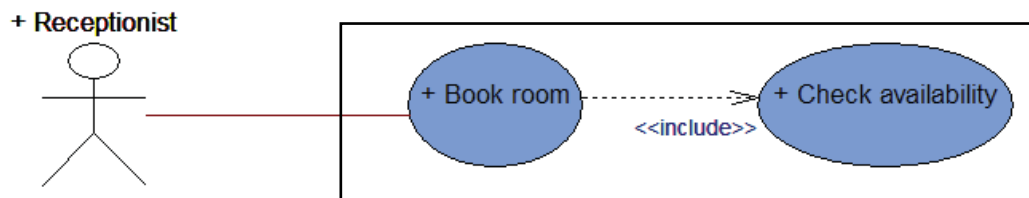
Fig 3.9 A use case with <<include >> association

Note. The use case symbols are shown enclosed within a box to indicate the boundary of the system and the lines linking the actors to use cases are called "**communication associations**". Note that some diagrams may include arrow tips pointing in the direction of the communication. An **<<extend>>** relationship can also be used to show the optional use of a use case, for example a use case "Make payment" might be extended to include a use case for "Credit card payment" and an alternative use case for "Cash payment".

Use cases should include all related transactions, so in the case of a guest booking into a hotel the associated transactions include room booking, issuing the bill and bill payment. This situation is shown by the use case diagram below.
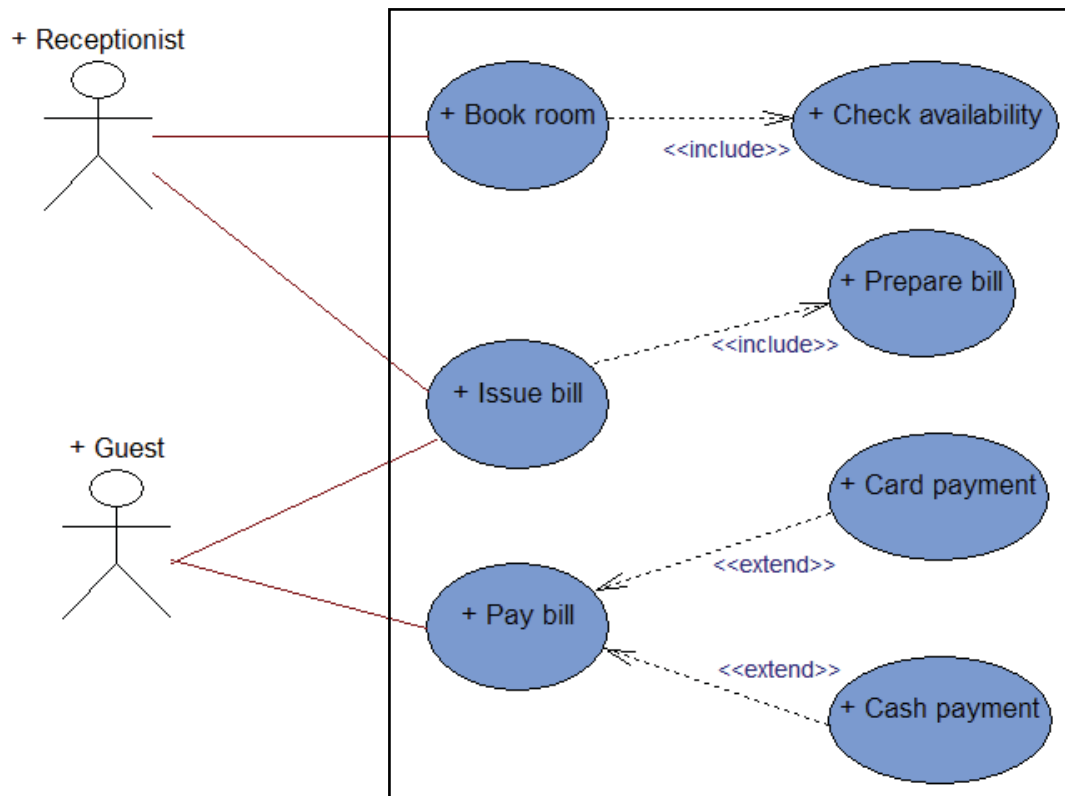
Fig 3.10 Hotel booking **use case** diagram

Exercise 3

Produce a use case description for the following scenario:-

"The hotel housekeeping manager checks the toiletries stock file on a weekly basis to ensure that there are sufficient quantities of soaps, shampoos etc. available. Where necessary, an order is raised to be sent to the supplier. Copies of the orders are kept and used to check the supplier delivery notes and invoices when they are received. Occasionally the suppliers are unable to send the requested toiletries and will suggest alternatives. Once the toiletries have been supplied the stock file is updated."

Exercise 3 feedback

Main success scenario

1. Check stock file weekly
2. If stock quantity < reorder level, raise supplier order
3. File order copy
4. Check order against delivery note/invoice
5. Update stock file with supplies.

Extensions

2a Supplier not able to supply toiletries

    2a1 Choose alternative toiletries

    2a2 Return to step 3 in the main success scenario.

3.3   CLASS DIAGRAM

When the analyst has identified the object classes from the **use case** diagrams, a logical model in the form of a **class diagram** is produced to show the associations between them. A **class association** is used to represent the relationship between two classes. It has a descriptive label and shows **multiplicity**. Multiplicity describes how many instances of one class in the relationship relates to instances of the other class. This is also sometimes referred to as **cardinality**.

The multiplicity of a relationship indicates the minimum and maximum number of occurrences and is shown using the following symbols:-

        0..1   is read as   zero or one

        1..1   is read as   one and only one

        0.. *   is read as   zero one or many

        1.. *   is read as   one or many

In the following class diagram the binary relationship between MODULE and LECTURER would be read from right to left as one LECTURER delivers one or many MODULEs (1..*) and read left to right, one module is delivered by zero or one lecturer (0..1). The implication here could be either that a module may not yet have a lecturer assigned to deliver it, or that it is a self-study module.

Now let us consider the many-to-many relationship between STUDENT and MODULE. As a student can study one or many modules (1..*) and a module may be studied by zero, one or many students (0..*) there will be a need to store additional data generated by this relationship – the assessment mark and grade for each student taking a module. Whilst you may consider storing this information in either of the classes, this would not be a practical solution due to the potentially large number of items of data, so the data is held in a new **association class** named STUDIES, which is shown linked to the initial relationship with a dashed line. This approach resolves the many-to-many relationship in an effective implementation that can cope with any number of students taking any number of modules.
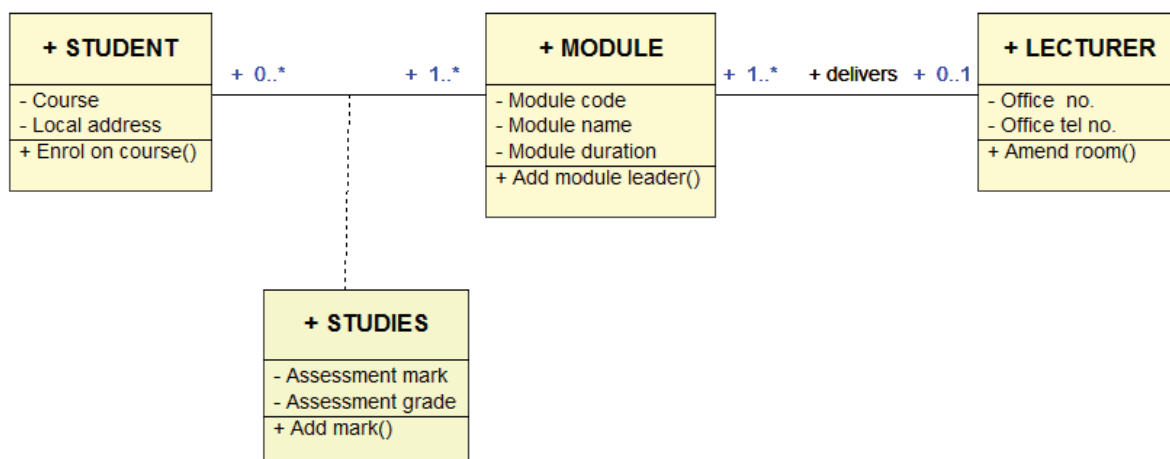
Fig 3.11 Example Class diagram

Note. It is important to consider each relationship from **both** directions to ensure you specify the correct multiplicity when drawing a class diagram.

3.4   SEQUENCE DIAGRAMS

The **sequence diagram** is used to show the interactions between objects in the sequence that they occur. They can prove useful in showing the dynamics of how a use case works and can be used to document the requirements for the new system. They are produced using the information acquired from the use case diagrams and scenario descriptions. Sequence diagrams can be drawn in varying degrees of detail depending on where they are being used within the SDLC. The basic elements are shown in the diagram below.

The following shows a sequence diagram for the Exercise 3 Toiletries stock check:-
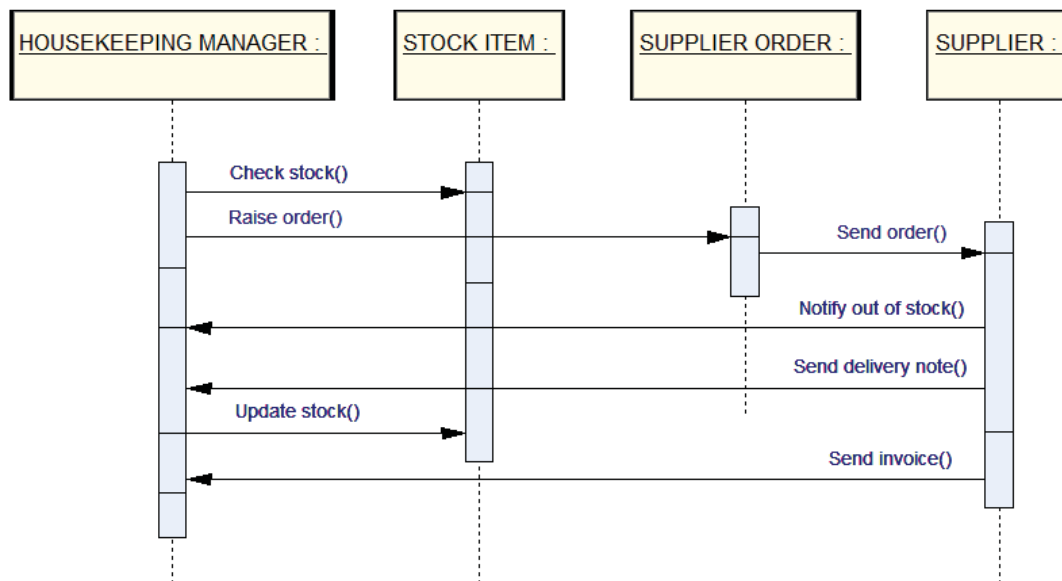
Fig 3.12 Simple sequence diagram for stock ordering

The sequence diagram is made up of four main components:- **classes**, **lifelines**, **messages** and **activation boxes**.

The classes HOUSEKEEPING MANAGER, STOCK ITEM, SUPPLIER ORDER and SUPPLIER are represented by rectangles.

The **classes** have lifelines which are represented by the vertical dashed lines. The lifeline indicates the time period that its class object is communicating with the other objects.

The **messages** are shown as horizontal arrowed lines pointing in the direction the message is travelling, and include a label indicating the name of the message. The first message is usually shown at the top with subsequent messages being arranged below one another. When messages are sent to an object this triggers a method (operation) within that object. Sequence numbers can be added to the messages, though this is not usually necessary as the vertical order of messages indicates the sequence.

The **activation boxes** are shown as a vertical bar (in blue) over the lifeline and this indicates the period of time when an object is carrying out an operation often referred to as the **focus of control**. A large **X** is sometimes shown at the bottom of an activation box to indicate the destruction of an object at the end of its lifeline.

Note. Complex interactions may require more than one sequence diagram to fully represent the main and alternative scenarios.

Sequence diagrams are useful when you have a complex scenario to work through.

3.5   STATE MACHINE DIAGRAMS

As has already been shown, objects have behaviours and states. In order to understand the various states an object goes through during its existence a **state machine diagram**, sometimes referred to as a **state transition diagram**, can be drawn. This diagram shows the transitions between the states that an object may undergo, caused by internal or external events. State machine diagrams tend to be created when examining a class with a complex life cycle.

The state machine diagram shown below represents a university student and the various stages or states that they transition. The round-edged rectangles represent the states:- initially the student will be regarded as an applicant, then they will be enrolled, when they arrive they will be registered and normally they will complete their course. Whilst at the university they may be withdrawn either temporarily or permanently. The small solid circle is the initial state, i.e. when the object starts its interaction with the system. The arrows indicate the event that triggers the transition from one state to another. The solid circle with the border represents the object's final state.
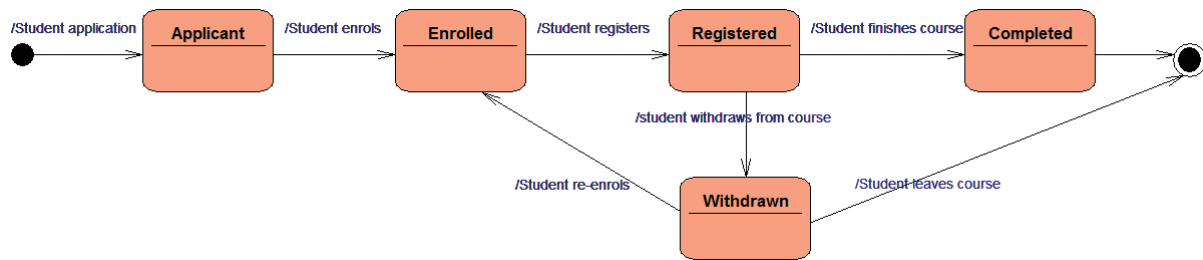
Fig 3.13 A state machine diagram for a university student.

3.6   ACTIVITY DIAGRAMS

An **activity diagram** is used to show actions and events in the order they take place, along with their outcomes. The following activity diagram is based on the scenario described in Exercise 3 above. The solid circle represents the start state of the diagram. The rounded rectangle represents an activity (manual or automated), the arrow represents an event (something that happens at a time and place) and the diamond represents a decision or branch, the condition value can be shown and is referred to as a guard condition. Additional symbols that can be used are a flat rectangle bar which represents a synchronisation, a fork which allows activities to take place in parallel and a join which brings events together. The final state (end of diagram) is shown by a solid circle with a border.
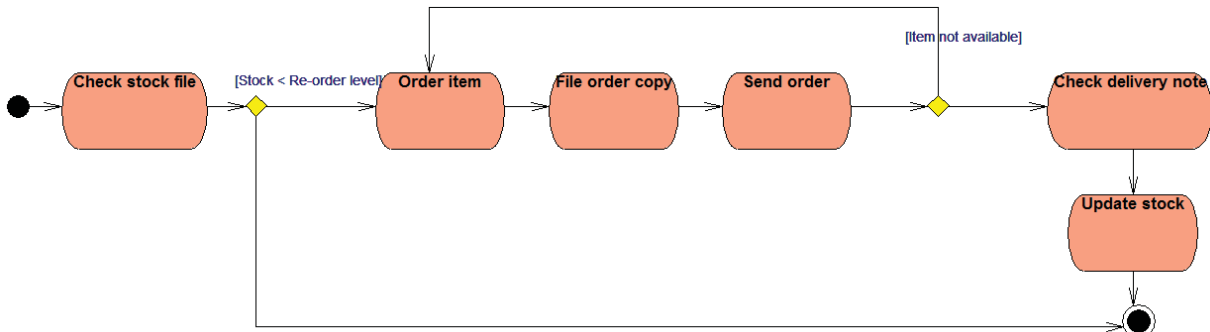
 Fig 3.14 An activity diagram for hotel stock ordering

Activity diagrams can be partitioned using pairs of lines referred to as **"swimlanes"** to show the activities which are performed by the different roles participating in the process. A role can be a user type – e.g. manager – or a platform – e.g. a web server. In the above scenario all the activities are carried out by the hotel housekeeping manager so swimlanes have not been included.

To create an activity diagram, consider the order in which activities take place and consider any conditions that may arise so that all the scenarios for the use case have been considered. If you have produced a physical DFD this may help to identify the order of activities.

3.7   BUSINESS PROCESS MODELLING

One final graphical modelling technique that can be used to represent how systems work, showing the business processes, events and people involved, is the standard **business process model and notation** (**BPMN**) (Object Management Group Business Process Model and Notation, 2014). The BPMN can be used to model requirements and can also be used in later stages of the development cycle. Its purpose is to provide a standard notation which can be understood by analysts, developers and other business stakeholders. A diagram consists of events, activities (tasks), sequence flows and gateways, which control the direction of flows.

The following simple business process diagram shows the hotel room booking processes mentioned in section 3.2 above.
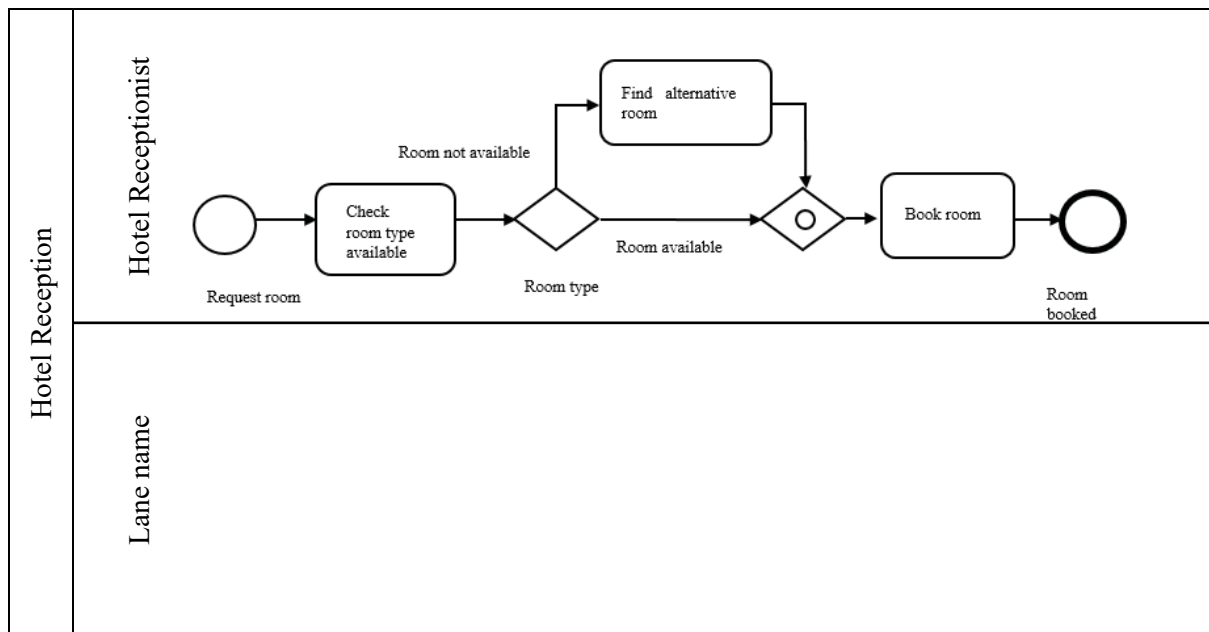
Fig 3.15 Simple business process diagram for room booking

"Request room" is the starting event. The first activity task "Check room type available" is carried out, followed by an exclusive gateway (diamond) which represents the "Room type" check to ascertain whether it is available; if not, another task "Find alternative room" is undertaken. This is followed by an inclusive gateway (diamond with a circle) which routes the sequence flow to the task "Book room".

The business process diagram can be partitioned to show which tasks are undertaken by different participants, although in this example all the tasks are undertaken by the hotel receptionist. A diagram typically consists of a pool (like a swimming pool, shown as a large rectangle positioned horizontally or vertically) which is partitioned into a number of lanes. Each lane shows a sub-partition of a process which is carried out by a participant. The above example only utilises one lane as the whole process is carried out by just one participant – the hotel receptionist. If further participants are involved these would appear in different swim lanes and the flows and symbols would be inserted in the relevant swim lane with flows linking across the swim lanes when necessary.

Detailed diagrams can be drawn utilising the full set of symbols which include messages, timers, signals and further gateway types. For the full specification see http://www.omg.org/spec/BPMN/2.0.2/ (Object Management Group Business Process Model and Notation, 2014)

3.8   SUMMARY

The analyst can use the UML to analyse the system requirements from an object-oriented perspective. This involves producing a set of diagrams to reflect the user's needs and specify the system's requirements. These include use case diagrams, class diagrams, sequence diagrams, state machine diagrams and activity diagrams. These should be reviewed with the business users until all are satisfied that they accurately reflect the requirements. The UML models will in effect provide a detailed set of specifications to be used later in the systems design phase of the SDLC. The O-O approach to modelling aims to utilise the reusability of objects and also aids maintainability of systems.

Further reading:

(Agile Models Distilled: Potential Artifacts for Agile Modeling, n.d.)

4   SYSTEMS DESIGN

On completion of this chapter you should be able to:

- produce an efficient data design
- design an effective user interface
- describe system architectures.

Following the completion of the systems analysis phase of the traditional SDLC, the system development phase can begin. However, at this point a decision may be made to consider available options for acquiring the system. Typical options include outsourcing, in which an outside organisation (a service provider) provides the system (as an application) and in some cases operates and maintains it (as an application service provider). If considering outsourcing an organisation needs to take account of issues such as security, levels of risk and cost – not just initial development costs but also the **total cost of ownership (TCO)** which includes the lifetime costs of the system. Outsourcing may be used just for the system development, as this may save the organisation the cost of acquiring additional staff who may not be needed on completion of the development.

Sometimes it may be necessary to decide whether to purchase a commercial package if one exists that meets or can be customised to meet the systems requirements, or whether to develop the system within the organisation, though this approach may still include outsourcing some of the development work. Typical reasons for in-house development are that the system produced would fully meet the requirements and allow for control of any technical or existing system constraints. Developing in-house may also offer greater security by avoiding the risk of the provider ceasing to offer support in the future. Reasons for purchasing an existing product usually relate to lower costs (mainly development), although the TCO may prove to be higher. Choosing an existing product with supplier updates that has a proven history may offer a more reliable and quicker implementation. The analyst will be involved in considering these acquisition options and needs to be able to evaluate the advantages and disadvantages in order to make a recommendation to the management in order to inform the final decision.

In traditional development environments the requirements specification is handed over to the software developers who will convert the logical models into a physical design. Physical design involves producing specifications that describe the data structures and the data validation rules, the business processes and the input and output formats for the system interface (i.e. screen forms and reports). When the design and development is completed the system is handed over to the users. This structured approach poses a risk that the delivered system may not satisfy the users even though it meets the agreed functional requirements. The alternative Agile development approach involves the users throughout the development phase by gaining their feedback on the design and development of small work packages at regular intervals. This approach aims to meet the needs of the users more closely and minimise any costly redevelopment changes.

4.1 DATA DESIGN

During the analysis phase the data requirements were identified, these requirements need to be structured in an efficient form so as to ensure data integrity and easy maintenance. Traditional legacy systems relied on separate data files, often resulting in duplicated data which caused update problems when an occurrence of a data value was changed in one file location but not the others. Most current information systems use a database for data storage as this allows data structures to be added or altered without needing to modify the systems using the data. **Relational database management systems (RDBMS)** allow an organisation to maintain all of its enterprise-wide data in an efficient form that provides flexible and secure data access. Note that some smaller systems may just make use of a local database system, but these are designed in the same way. Databases may be accessed remotely via the Internet or internally via a local area network (LAN).

Data within a database is accessed and maintained using a **data manipulation language (DML)** An example of this is the widely used **structured query language (SQL)**. **Query by example** graphical front end tools which allow users easier access to the data they require are also available with some databases. The data within a database is stored in **tables** which consist of **rows** representing data records, e.g. a student record. Each row comprises a specified number of columns (often called fields) for storing individual data values, e.g. student name, address, telephone number etc. Each column has a defined data type. Each row is identified by a **primary key (PK)**; this makes it possible to retrieve an individual row (record) from within the table. A primary key is a unique value which is contained within one column, although in some cases a compound key may be formed by using more than one column.

The tables are structured in a simple form to eliminate data duplication resulting in data being stored across a number of tables within the database. In order to extract the required related data to satisfy an information requirement, tables need to be linked. This is achieved by including a **foreign key (FK)** column in a table which holds a value that matches a primary key column value in the relevant table. To illustrate this, consider a university scenario where you have the database tables STUDENT, COURSE and TUTOR. In order to identify the course a student is studying, the two tables STUDENT and COURSE need to be linked. This is achieved by including a foreign key column called Course_Id in the STUDENT table. Likewise, in order to identify a student's personal tutor, the personal tutor's id is stored as a foreign key within the STUDENT table, which provides a link to the TUTOR table primary key column Tutor_Id as shown below:

Table: STUDENT

Student_Id (PK)	Student_name	Course_Id (FK)	Personal_Tutor (FK)
S10102	M Jones	C101	L1111
S10103	L Shaw	C101	L1111
S10104	A Khan	C107	L1203

Table: COURSE

Course_Id (PK)	Course_name	Level
C101	BSc. Computing	UG
C102	BSc. Information Systems	UG
C107	MSc. Software Engineering	PG

Table: TUTOR

Tutor_Id (PK)	Tutor_name	Tutor_tel_no.
L1111	Dr Dixon	778907
L1202	Dr Jones	779967
L1203	Dr Berger	775565

Fig 4.1 Tables with primary and foreign keys

The database system maintains **referential integrity** automatically; this means that it will ensure that foreign key values match with an existing primary key value in the referenced table. Likewise, duplicate primary key values in a table are prevented.

4.2   ENTITY MODELLING

In order to design the physical database table structures needed for the system, further analysis of the data requirements as identified from the systems analysis phase must be undertaken. This is achieved by using **Entity modelling** which is a top-down approach that involves producing an **entity relationship diagram (ERD)** showing the data **"entities"** and their **"relationships"**. A data entity is formally called an **"entity type"** which represents a data object relevant to the system and can be defined as follows:

> 'A group of objects with the same properties which are identified by the enterprise as having an independent existence" (Connolly, 2015, p. 406)

An ERD uses the following symbol to represent an entity. Each entity is given a name which is always shown in the singular and in upper case text e.g.
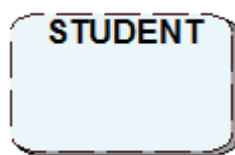
Fig 4.2 Entity type

Entity types are linked by means of **"relationships"**. The relationships are shown as lines with descriptive labels, linking the entity types on the ERD. The relationship lines include additional symbols which represent the **"cardinality"**. Cardinality defines the number of occurrences of an entity type as it relates to one occurrence of the other entity type in the relationship. The cardinality symbols used give rise to the notation name **"crow's foot"** as the "many" symbol looks like a crow's foot.

There are three main types of relationship:-

One to one relationship (Abbreviated as 1:1)

Fig 4.3 A one to one relationship

In a one to one relationship, one occurrence of an entity type *only* relates to one occurrence of the other entity type. The above example represents the situation in which a lecturer acts as a course leader for one course and a course only has one course leader (lecturer). This would be read as "a lecturer leads one and only one course and a course is led by one and only one lecturer".

One to many relationship (Abbreviated as 1:M)
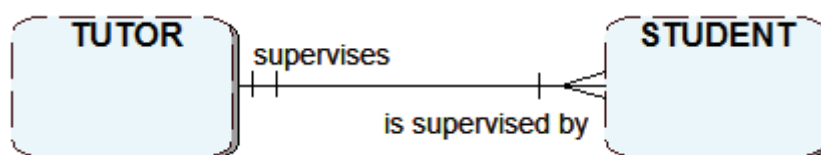
Fig 4.4 A one to many relationship

A one to many relationship is where one occurrence of an entity type relates to at least one and possibly many occurrences of the other entity type, as in the above example where one tutor may supervise one or more students. The two vertical lines against the TUTOR entity type indicate that an occurrence of the TUTOR entity type is mandatory, i.e. must be present in the relationship. The vertical line and crow's foot symbol against the STUDENT entity type indicates that at least one, but possibly many, occurrences of the STUDENT entity type can be present. The above relationship would be read as "a tutor supervises one or many students and a student is supervised by one and only one tutor".

Many to many relationship (Abbreviated as M:M or M:N)
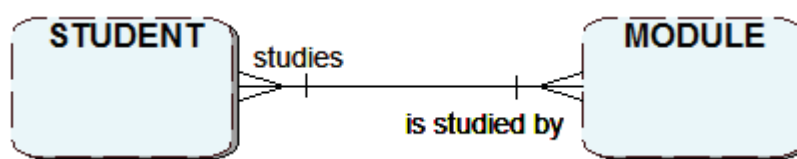
Fig 4.5 A many to many relationship

In a many to many relationship, an occurrence of one entity type will relate to one or many occurrences of the other entity type and vice a versa. The above example would be read as "a student studies one or many modules and a module is studied by one or many students". Note the abbreviation M:N is often used as this indicates that the number of occurrences at one end of the relationship may be different from the number of occurrences at the other end of the relationship.

In some relationships an occurrence of an entity type may not always be present. In this situation the relationship is said to be **"optional"**. To illustrate this, consider the relationship shown in Fig 4.4 above. This shows that a tutor supervises at least one student, however what if some tutors do not supervise students? This optionality can be shown on the relationship, denoted by a small circle on the relationship line against the entity type that is optionally present:-
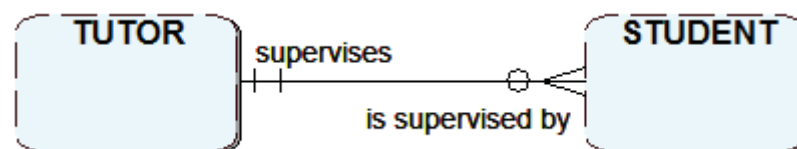
Fig 4.6 An optional one to many relationship

The relationship now reads as "a tutor supervises zero, one or many students and a student is supervised by one and only one tutor". The zero implies optionality. Optionality may be present at both ends of a relationship. For example, using Fig 4.5 above, if a student can exist who has not studied any modules, a circle could be placed against the module entity type, and if a module may not be studied by any students i.e. it is a new module which is not yet being taught, this can be represented as follows:-
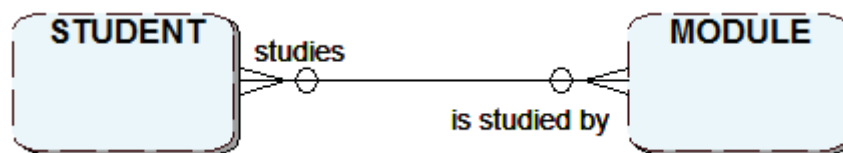
Fig 4.7 A many to many relationship with optionality

Referring to Fig 4.3 above, if you needed to show that not all lecturers lead courses then optionality would be shown against the course entity type as shown below. This would be read as "a lecturer leads zero or one course and a course is led by one and only one lecturer":-

Fig 4.8 A one to one relationship with optionality

Note. It is important to read a relationship in both directions to ensure that the correct cardinality and optionality are specified, as mistakes will affect the subsequent table implementation. You can think of the inner relationship symbol (| or O) as the minimum number (one or zero) of occurrences and the outer symbol (| or <) as the maximum number of occurrences (one or many).

The many to many relationship type differs from the other types in that the relationship itself holds data which cannot be stored effectively in either of the participating entity types. Let us consider the M:N relationship shown in Fig 4.7 above. A student may study many modules and a module may be studied by many students. This implies the need to be able to identify which modules a particular student is studying and also which students are studying a particular module. It would not be practical to store each module studied by a student in the student entity; likewise, it would not be practical to store every student studying a module in the module entity. This situation is resolved by replacing the M:N relationship with a new entity type referred to as an **"associative entity"** (often called a link entity) which is linked with 1:M relationships to the other two entity types.

An occurrence of the STUDY MODULE associative entity type would include two columns, one to hold the primary key value of a student (Student_Id) and the other would hold the primary key value of the module (Module_Id). Note that these two values act as foreign keys within the associative entity type and when used together form a compound primary key. This approach allows for any number of students studying any number of modules to be recorded easily. This altered relationship is shown as follows:-
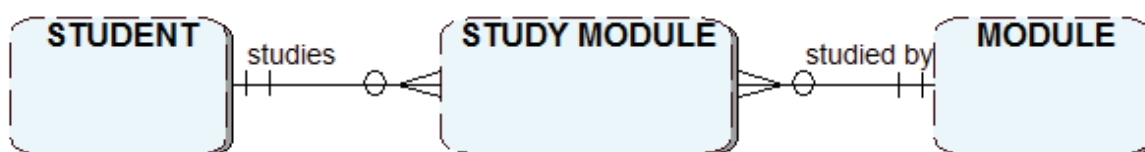
Fig 4.9 An associative entity

The above entity relationships can now be combined to form a complete entity relationship diagram:-
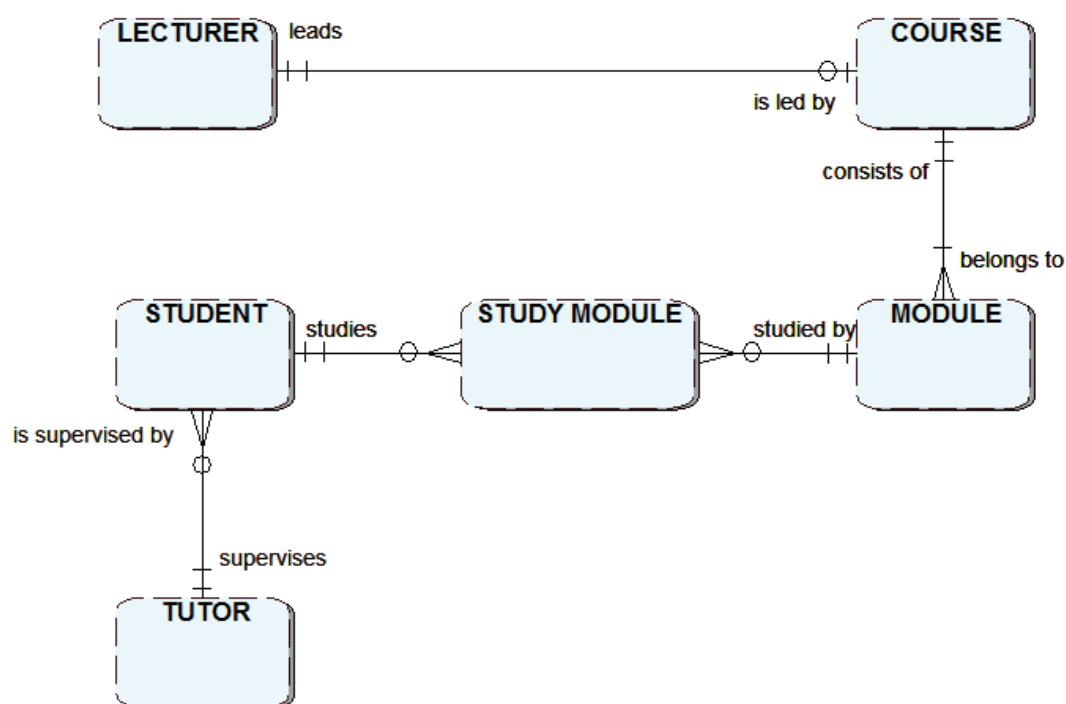
Fig 4.10 University ERD

More complex relationships can be modelled using extended modelling techniques. These are not covered in this text; readers are encouraged to refer to the author's book Database Design and Implementation (Gould, 2015) for a more detailed explanation of modelling and database design.

A set of **"relations"** can be derived from the ERD with reference to system data identified during the analysis phase. A relation, not to be confused with a relationship, is a logical representation of an entity type and consists of its **"attributes"**. The relations are similar to physical data tables and the attributes are similar to table column values. A relation can be represented as follows:-

Relation name (attribute1, attribute 2, etc.)

e.g. STUDENT (**<u>Student_id</u>**, student_name, student_DOB, *Tutor_id*)

Note. The Student_id is shown in **bold** and <u>underlined</u> to signify that it is a unique identifier (primary key), foreign key attributes, if present, are shown in *italics.*

Initially, when deriving relations not all attributes may be known so '**skeleton relations**' are produced which only include the primary and foreign key attributes.

4.3   NORMALISATION

Producing relations involves identifying all the data attributes that relate to a specific entity type, including any foreign key values which can be identified by considering the entity relationships. When the relations have been produced a set of rules is applied to each one to ensure that they are in a form that avoids data redundancy and results in efficient table structures. This process is called **"normalisation"** and is often referred to as a bottom-up design approach, as it focusses on grouping the data attributes together to form appropriate relations. Data source documents (forms/reports) may also be normalised directly in order to identify their relations, some of which may not be present on the ERD but need to be included. Following this process, the table design can be finalised.

At the logical design stage the relations may be in one of the following states:-

- Un-normalised (UNF),
- 1st Normal Form (1NF),
- 2nd Normal Form (2NF),
- 3rd Normal Form (3NF).

By applying the rules, a relation is transformed until it is in 3rd normal form, which is commonly referred to as **"normalised"**.

An un-normalised relation contains a repeating group of attributes. Any repeating group needs to be separated from the original relation and placed into a new relation with its own primary key. The identifying attribute for the new relation remains in the original relation to act as a foreign key to provide a link to the new relation. This process can be illustrated as follows:

The following example shows an un-normalised table called **STUDENT**:

Student Id	Student Name	Module Id	Module name	Module Grade	Course Id	Course name
101	J Smith	DB	Databases	75	CMP	Computing
101	J Smith	SA	Systems Analysis	65	CMP	Computing
101	J Smith	PR	Programming	70	CMP	Computing
102	A Khan	DB	Databases	55	CMP	Computing
102	A Khan	PR	Programming	60	CMP	Computing
103	R Berger	DB	Databases	50	BIT	Business IT
103	R Berger	WD	Web Development	55	BIT	Business IT
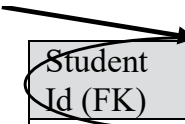
Fig 4.11 Un-normalised table

The attributes Module Id, Module name and Grade are repeated for each student, so to transform the STUDENT table into 1NF these repeating attributes are removed and placed into a new table called MODULE GRADE. The primary key for the new table will be a compound key consisting of the Module Id and the student Id (also a foreign key). This results in the following 1NF tables:-

STUDENT

Student Id (PK)	Student Name	Course Id	Course name
101	J Smith	CMP	Computing
102	A Khan	CMP	Computing
103	R Berger	BIT	Business IT

Compound Primary key

MODULE GRADE

Student Id (FK)	Module Id	Module name	Module Grade
101	DB	Databases	75
101	SA	Systems Analysis	65
101	PR	Programming	70
102	DB	Databases	55
102	PR	Programming	60
103	DB	Databases	50
103	WD	Web Development	55

Fig 4.12 1NF tables

To transform 1NF tables into 2NF involves checking for part-key dependencies and removing these to form a new table. To explain what a part key dependency is, let us look at the 1NF MODULE GRADE table in Fig 4.12.

This table has a multi-part key consisting of the Student Id and the Module Id. A check is made to identify if any other columns are dependent on only one part of the key (i.e. determined by a part of the key). On inspection, it becomes apparent that the Module name can be determined by just the Module Id, so there is a part key dependency. This results in the Module Id and Module name being used to form a new 2NF table called MODULE. Contrast this with the module grade column; this can only be determined by having both the Student Id and the Module Id, so is dependent on the full key and remains in the MODULE GRADE table.

As the MODULE table only has a single column key – Module Id – the table is, by default, already in 2NF. The 2NF tables are as follows:-

STUDENT

Student Id (PK)	Student Name	Course Id	Course name
101	J Smith	CMP	Computing
102	A Khan	CMP	Computing
103	R Berger	BIT	Business IT
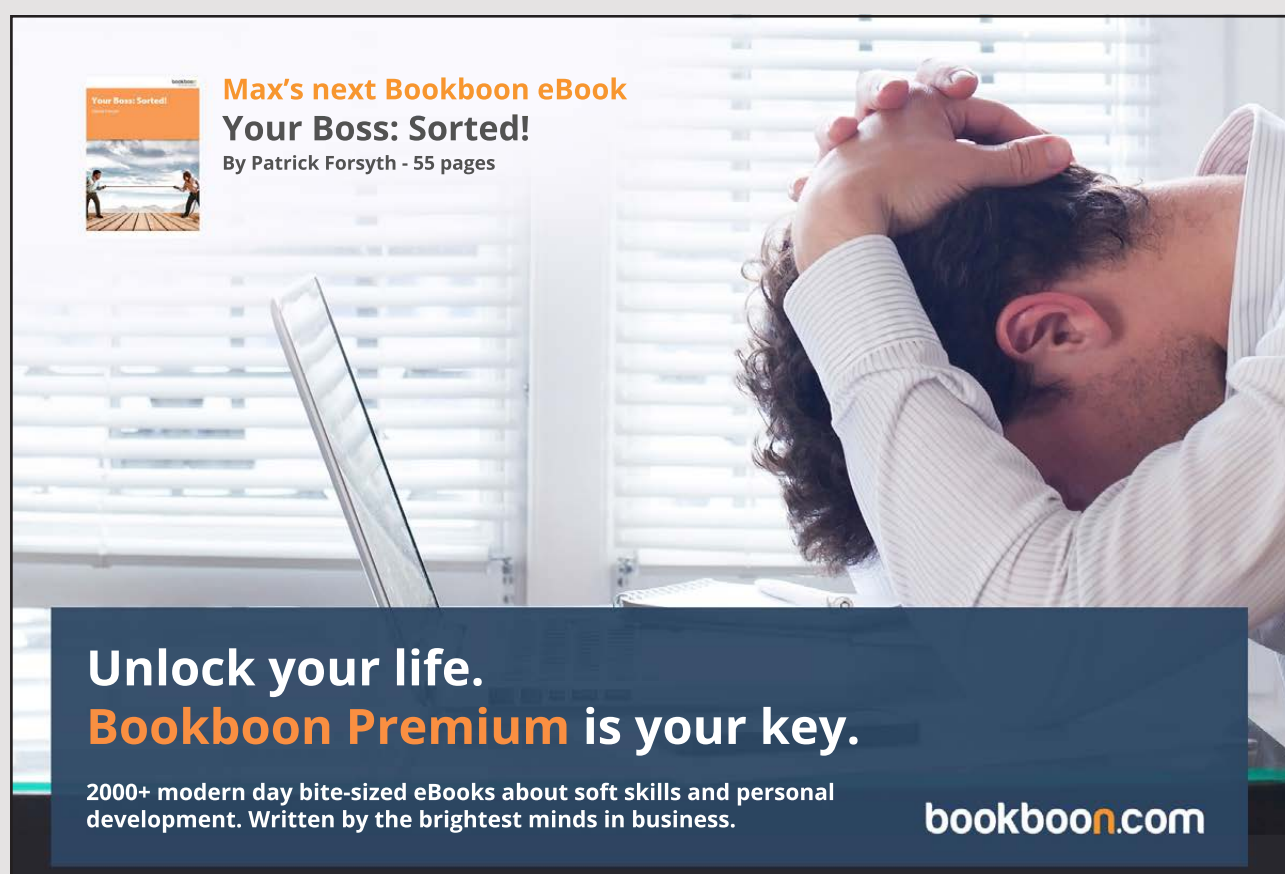
MODULE GRADE

Student Id	Module Id	Module Grade
101	DB	75
101	SA	65
101	PR	70
102	DB	55
102	PR	60
103	DB	50
103	WD	55

MODULE

Module Id (PK)	Module name
DB	Databases
SA	Systems Analysis
PR	Programming
WD	Web Development

Fig 4.13 2NF tables

The final step is to ensure that the tables are in 3NF. This is to ensure that all non-key attributes are dependent on the key and only the key. To illustrate this, let us look at the 2NF STUDENT table from Fig 4.13 above.

The Student name and Course Id are determined by the key Student Id, however the Course name can be determined by the non-key attribute Course Id, so the course name can be placed in a separate table called COURSE, along with the Course Id which will be the table primary key. The other 2NF tables are already in 3NF so they remain unchanged. This results in the following normalised set of tables:

STUDENT

Student Id (PK)	Student Name	Course Id (FK)
101	J Smith	CMP
102	A Khan	CMP
103	R Berger	BIT

MODULE GRADE

Student Id	Module Id (FK)	Module Grade
101	DB	75
101	SA	65
101	PR	70
102	DB	55
102	PR	60
103	DB	50
103	WD	55

MODULE

Module Id (PK)	Module name
DB	Databases
SA	Systems Analysis
PR	Programming
WD	Web Development

COURSE

Course Id (PK)	Course name
CMP	Computing
CMP	Computing
BIT	Business IT

Fig 4.14 3NF tables

Note. In some situations, a decision may be made to denormalise the tables for performance reasons but this should be avoided if at all possible.

> See Appendix B Normalisation template which will assist you with the normalisation process.

> Remember attributes in a relation should depend on:
>
> **1NF – the key**
>
> **2NF – the whole key**
>
> **3NF – nothing but the key**

The set of tables from Fig 4.14 can be represented by the following ERD:-
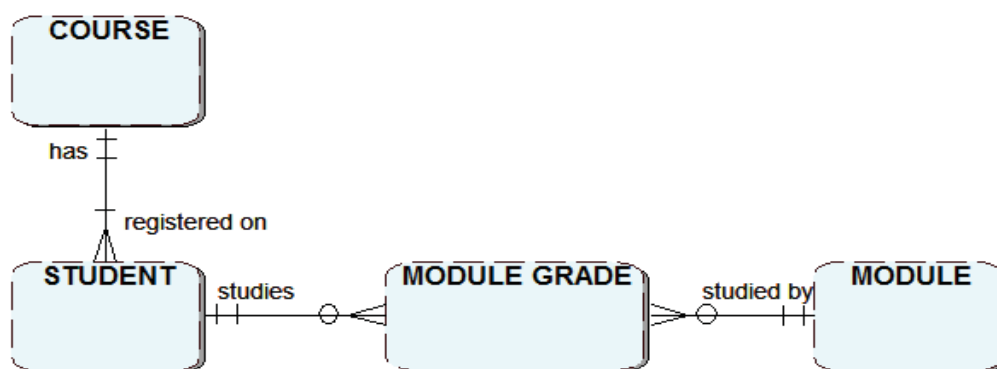
Fig 4.15 Student ERD

4.4   IDENTIFYING RELATIONS

The rules for deriving the relations and their keys from an ERD are summarised as follows:

For a 1:M relationship the Primary Key from the 'one' end of the relationship is included at the 'many' end of the relationship as a foreign key, e.g. the primary key Course_id of COURSE is stored in STUDENT as a foreign key. Optionality at the 'many' end does not affect this rule but optionality at the 'one' end does.

If there is optionality at the 'one' end of the 1:M relationship, a new relation is needed, as this avoids a NULL (empty) foreign key value being stored at the 'many' end of the relationship. **Note.** NULL keys are to be avoided as they would not identify any row within the database table.

The new relation will hold the primary key from the 'many' end of the relationship as the foreign and primary key in the new relation, which will also contain the primary key from the 'one' end of the relationship as a foreign key. This is shown as follows:-
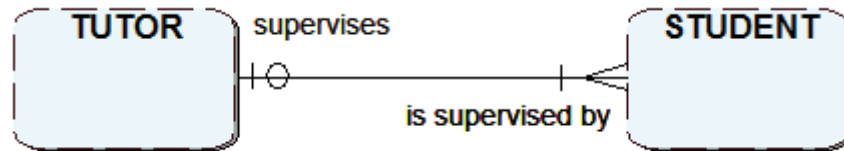
TUTOR(**Tutor_Id**)   STUDENT(**Student_Id**)   TUTEE(*Student_Id*, *Tutor_Id*)

Fig 4.16 1:M relationship with optionality

For a 1:1 relationship the primary key from either end of the relationship may be deposited as a foreign key at the opposite end of the relationship. **Note.** Often a 1:1 relationship without optionality gives rise to a single table containing all the attributes from each entity with either key being used as the primary key.

If the 1:1 relationship includes optionality at just one end of the relationship, the primary key from the non-optional end is inserted as the foreign key at the optional end of the relationship.

If the 1:1 relationship includes optionality at both ends, a new relation is needed to avoid NULL foreign keys. The new relation contains both the primary keys as foreign keys, and either of the foreign keys can be used as the primary key in the new relation.

For a M:N relationship a new relation is created which contains the primary keys from each of the original relations as foreign keys; both together are used to form a compound primary key.

Exercise 1

Produce a set of relations in third normal form (3NF) for the following Invoice document (Gould, 2015, p. 70), Use the Appendix B normalisation template:-

---

Yorkshire Computer Supplies

1 Long Road, Leeds, LS3 3QS, West Yorkshire, UK. Tel: 0113 2832700

INVOICE

Invoice No: 1034

Invoice Date: 31/1/2015

Customer No: C101

Invoice Name / Address

H. Jones

9 The Avenue

Harrogate

HG2 7LR

ITEM ID	DESCRIPTION	QTY	PRICE	AMOUNT
PC1	Computer	3	500.00	1500.00
MN2	Monitor	3	200.00	600.00
LP1	Printer	1	156.00	156.00
		SUBTOTAL		2256.00
		TAX @ 20%		451.20
		DELIVERY		35.00
		TOTAL		2742.20

Exercise 1 feedback

UNF	1NF	2NF	3NF	Relation / Table Name
1. List all the attributes below from a single document / table. 2. Identify the unique identifier / primary key. Show in bold or colour. May need an artificial key. 3. Identify any repeating attribute group(s). Show between (….) or use colour.	1. Place repeated attribute group(s) if any in a new relation. 2. Include the UNF unique identifier as a foreign key in the new relation. 3. Identify the additional attribute(s) in the new relation to form a compound key with the foreign key.	1. Remove any part key dependent attributes to a new relation. 2. Identify identifier for each new relation. 3. Include foreign key in the original relation.	1. Remove any non-key dependent attributes to a new relation(s). 2. Identify the unique identifier for the new relation(s). 3. Include a foreign key in the original relation.	Assign a suitable name for each relation/table.
Inv_no	**Inv_no**	**Inv_no**	**Inv_no**	**INVOICE**
Inv_date	Inv_date	Inv_date	Inv_date	
Inv_customer_no	Inv_customer_no	Inv_customer_no	*Inv_customer_no*	
Inv_name_address	Inv_name_address	Inv_name_address	Inv_sub_total	
Inv_item_id	Inv_sub_total	Inv_sub_total	Inv_tax	
Inv_item_desc	Inv_tax	Inv_tax	Inv_delivery	
Inv_item_qty	Inv_delivery	Inv_delivery	Inv_total	
Inv_item_price	Inv_total	Inv_total		
Inv_item_amount			**Inv_customer_no**	**CUSTOMER**
Inv_sub_total	*Inv_no*	*Inv_no*	Inv_name_address	
Inv_tax	**Inv_item_id**	*Inv_item_id*		
Inv_delivery	Inv_item_desc	Inv_item_qty	*Inv_no*	**INVOICE ITEM**
Inv_total	Inv_item_qty	Inv_item_amount	*Inv_item_id*	
	Inv_item_price		Inv_item_qty	
	Inv_item_amount	**Inv_item_id**	Inv_item_amount	
		Inv_item_desc		
		Inv_item_price	**Inv_item_id**	**ITEM**
			Inv_item_desc	
			Inv_item_price	

Note the Primary keys are shown in **bold** and the foreign keys are in *italics*. INVOICE_ITEM has a compound primary key

INVOICE (**Inv_no**, Inv_date, *Inv_customer_no*, Inv_sub_total, Inv_tax, Inv_delivery, Inv_total)

CUSTOMER (**Inv_customer_no**, Inv_name_address)

INVOICE_ITEM (**Inv_no**, *Inv_item_id,* Inv_item_qty, Inv_item_amount)

ITEM (**Inv_item_id**, Inv_item_desc, Inv_item_price)

4.5 DATA TABLE STRUCTURES

Once a set of relations has been derived, the final stage of data design involves specifying the physical table structures. This includes defining the columns and their data types and sizes. At this point the analyst may choose to use codes to identify some data values, e.g. student_id – S123456, course_code – CMP etc. Using codes – which usually consist of letters and/or numbers – appropriately can minimise data entry time, reduce data entry mistakes and save storage space. Where numeric-only codes are used to uniquely identify data records these are usually automatically generated by the system to avoid duplicates being entered in error.

Typical available data types include the following:-

Number – used to hold values which are used in calculations, e.g. hourly pay rate, are normally defined as **integers** if only whole numbers need to be stored, e.g. 121, or **real numbers** if decimal places are needed e.g. 5.25.

Character – used to store character strings of a fixed length which can be made up of letters, digits or other symbols commonly found on a computer keyboard. **Note**. Columns that do not contain the maximum number of characters specified will normally be padded out with spaces. Fixed size codes such as student id or items such as telephone numbers are usually defined as character data types.

Variable length character – used to store variable length character strings, these allow for larger strings to be stored but if a string that is less than the maximum specified length is entered, the data value is not padded with spaces and the system shrinks the column in order to save storage space.

Date – the date datatype is used when actual dates need to be stored. They are encoded so that they can be used in date checking arithmetic, whereas dates stored within a character field may look like dates when displayed, but are not stored as true dates therefore they cannot be recognised as such in date checking validation or calculations.

Note. The specific data types available and their storage formats will depend on the database management system specified for implementation.

> Care should be taken when specifying data types and sizes to ensure that they will hold the maximum required numerical values, character string lengths or date formats.

Similar data type principles also apply to conventional data files that may be used in some systems.

The following shows the ORACLE Structured Query Language (SQL) table structure definitions for the student ERD example shown in Fig 4.15 above:-

CREATE TABLE course (
Course_Id   char(3) Primary key,
Course_name   varchar(30));

CREATE TABLE student (
Student_Id   char(3) Primary Key,
Student_name   varchar2(30),
Course_Id   char(3) references Course (Course_Id));

```
CREATE TABLE module (
Module_Id          char(2) Primary Key,
Module_name        varchar2(30));

CREATE TABLE module_grade (
Student_Id         char(3),
Module_Id          char(2) references module (Module_Id),
Module_Grade       Integer,
Primary key (Student_Id, Module_Id));
```

The 'char 'datatype defines a fixed length character string and the value in ( ) represents the maximum number of characters that can be stored.

Varchar2 data type defines a variable length character string and the value in ( ) represents the maximum number of characters that can be stored.

Integer data type defines a column that can hold a value in the range -2147483648 to 2147483647.

SQL can be used to extract and manipulate data from within the database tables. The following is an example for the query "which students are studying the Databases module":-

```
SELECT Student_name
FROM student, module_grade, module
WHERE student.Student_Id = module_grade.Student_Id
      AND module_grade.Module_Id = module.Module_Id
           AND module.module_name = 'Databases';
```

Note that the data types and sizes are not specified within the SQL SELECT statement to extract the data, as the database maintains the stored table definitions.

Finally, data access and security need to be considered to ensure that users of the system are only granted access to data they are permitted to use. This is usually enabled by means of a system user access ID and password. Often, stored data is encrypted to prevent unauthorised access and audit logs may be kept to record information about a user's data access. The database administrator will be responsible for setting access rights and ensuring that the database is backed up regularly to ensure that in the event of a serious problem, previously backed up data can be retrieved and reinstated.

4.6   HUMAN-COMPUTER INTERACTION

Once the data requirements for the system have been identified, attention can be given to designing the user interface which will be passed to the system developers later. The user interface provides a means of communication between the system and the user and this is referred to as **human-computer interaction (HCI)**. It is vitally important to ensure **usability** – designing an effective **user interface (UI)** which allows the users to feel comfortable and to ensure they are as productive as possible when using the system.

A significant amount of research has been undertaken to identify good principles of UI design. One such example is "The Eight Golden Rules of Interface Design."

http://www.cs.umd.edu/~ben/goldenrules.html (Shneiderman B., 2010).

Most systems interact with users by means of a computer screen, keyboard and mouse utilising a **graphical user interface (GUI)** but some allow for voice or touch input and speech output. Other specialised devices may be used for speedier data entry such as **barcode scanners** or **radio-frequency identification (RFID)** tag readers.

It is important to gain early feedback on the UI design from users, and in order to facilitate this **storyboards** may be constructed; these are sketches which show the basic screen layouts. Software prototypes may also be constructed which simulate the actual screens. A UI may undergo a number of changes following prototype feedback and further changes may be made after the system goes live, as in some cases an interface may be assessed for performance by monitoring actual user usage and collecting usability metrics. Ultimately it is the system users who decide if the interface satisfies their needs.

The aim should be to produce a self-explanatory interface that is quick and easy to use, as most users do not like reading user guides or manuals.

The following should be considered when designing the interface:-

- All screens should be easily accessible, usually by means of a system main menu consisting of control buttons, menu bar tabs or navigation tree lists which, when selected, will take the user to a sub-menu or directly to a particular function screen.

  The following storyboard example shows a simple main menu screen for a student information system. If a user clicks on a control button labelled Students, Courses or Timetables they will be directed to a sub-menu screen which gives them further subject category choices. The Help button will provide general information about the system and the Exit button will allow the user to sign out of the system.
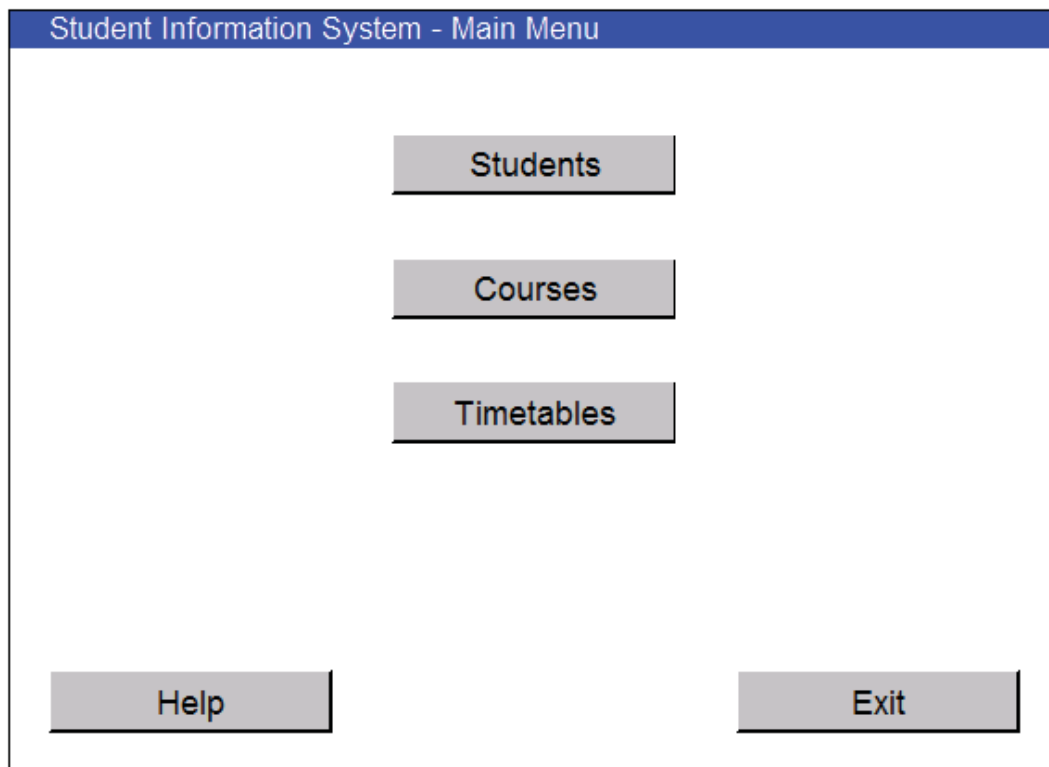
Fig 4.17 System main menu screen produced with QSEE storyboard tool

- Source documents that contain data to be input may influence the design of input screens.
- It should be possible to return to the main menu or sub-menus from any screen.
- Screen layouts should be consistent. This means that their look and actions should be the same throughout the system. For example, action buttons or menu tabs such as Save, Delete, Exit should appear in the same location on all screens that use them.
- Do not overload screens with too many items (data entry fields, lists, buttons etc.).
- Each screen should have a meaningful title indicating its main function.
- Colour schemes (background and text), typefaces and font sizes should be chosen to ensure readability. Remember to consider any users who are colour blind or have poor vision. Appropriate use of colours or sounds can highlight important information but take care not to overload users with too many of these.
- All control/command buttons, data fields and icons should be clearly labelled. Symbols or images used for buttons should be obvious.
- If commands or options are not available to a user, they should be hidden or greyed out.
- To ensure data quality, validation and error checking are included where possible to prevent invalid data from being entered. The common IT phrase "**garbage in, garbage out**" **(GIGO)** describes the likely outcome if this is not in place.

The following ORACLE APEX data entry and retrieval screen shows a typical master detail relationship form-based layout for displaying music companies (master record) and their CDs (related detail records), which incorporates some typical GUI options – radio buttons, drop down lists and check boxes. These help to speed up data entry and retrieval. Individual data field items can also be colour coded to highlight different types of data e.g. grey for not editable or red for an invalid entry value.

Fig 4.18 Master detail form screen layout

Typical checks that can be incorporated to ensure valid data is entered include:-

- **Data type** checks i.e. making sure only digits have been entered where a numeric value is required.
- **Input masks** which ensure that only data that is entered in a specified format can be accepted, e.g. a date in DD/MM/YY format.
- **Look-up checks** can be used to check a value exists, e.g. that a product code exists in the product file.
- **Range checks** can be used to ensure that a value entered is within an allowable range, e.g. between minimum and maximum hourly rates of pay.
- **Cross checks** can be made to ensure that the value in one field is compatible with the value in another field. For example, if a new student must be aged 18 or over to study a particular course the date of birth and the course code would need to be checked for compatibility.
- **Batch checks** can be used where batches of data are input together. A check can be made to ensure that all the items in a batch of source documents have been entered by totalling up a particular set of values in the batch then checking whether the system total is the same after the same values have been entered. A typical example of this would be totalling the order quantities on a batch of orders.

If data entry errors are detected the user should be prompted with an informative error message advising them of the problem and how to rectify the situation, preferably one which avoids entering all but the incorrect values again.

As it is important to be able to enter and retrieve accurate information quickly and easily and avoid entering data unnecessarily, the following guidance should be applied where possible:

- Use default values e.g. the system date for date fields or a date picker, and drop-down lists of items or automatic searching based on text as it is entered to select a value. This speeds up data entry and helps avoid typing errors.
- Group related data entry items together and ensure that the cursor tabs through them in the sequence needed for entry.
- Include short cuts where possible to speed up access to a menu or data entry, e.g. pressing the Ctrl and C keys to copy some text and Ctrl and V to paste it into a data entry field.

Feedback and help are important to users. Feedback ensures that the user is informed of their progress, i.e. has a data record been successfully saved or deleted. Help should be included to assist with general understanding and specific guidance relating to a particular action or request.

- Feedback messages should be clear and remain visible long enough to be read by the user.
- The user may be prompted to acknowledge they have read the message before it is removed, often by clicking on an "OK" button.
- Messages should be displayed in a consistent location on the screens so the users know where to look.
- Help should be available in a form that is relevant at the point it is requested.
- A general help facility may allow the user to find out information about a particular screen or function. Context sensitive help may be invoked when a user needs to know what needs to be done at a particular point in time, e.g. what needs to be entered into the data entry field in which the cursor is currently located.

Most systems include some reports that list or summarise data, in some cases in the form of graphs or charts. Traditionally, reports were printed and whilst some still are, the trend is towards screen-based reporting, although both forms share a number of design considerations.

Many systems now offer users the ability to specify or customise their own reports to show only the data required and in a format that they require for a specific purpose. In order to achieve this, they may use a report generator or wizard or alternatively a Query By Example (QBE) function. A QBE will allow the user to choose only the data values they want and then automatically generate an SQL SELECT statement or other program code to retrieve the required data from within the system database. As with screen design, users should be involved in the design process to ensure that the reports satisfy their requirements. This should include using report templates or mock-ups in order to gain feedback.

There are three main types of report:-

Detail reports – these list lines of output for each record and can often result in lengthy reports running to many pages e.g. lists of all customers or stock items. Consider carefully whether these reports are necessary, especially if they are printed, as this may prove costly if they need to be reprinted regularly due to frequent changes to the data.

Download free eBooks at bookboon.com

Summary reports – these tend to be used by managers and summarise the data that may appear on detail reports, for example a stock summary report may only list the stock item categories and the total number of items within the category, rather than every stock item. Another example might be a university that wants to know how many students it has on each course. In this case it is only necessary to list the total numbers.

Exception reports – are produced that show only details that match a specific criterion, for example all students who have been absent in the last week.

Report layouts should include the following:-

Report headers and footers. Headers appear at the start of the report and should include the report title and any relevant dates i.e. For Week 1 01/01/2016–07/01/2016. The header or footer should also include the date and time the report was produced to allow users to check they are looking at the correct version. Report footers tend to contain overall report or grand totals.

As reports may run to many pages each report should also have a **page header and footer** on every page. The page headers include column headings which should be concise descriptions of the data that will be displayed beneath them. The page footer will normally include the page number and report run date.

When reports include data grouping, a control break is used to show the totals for each individual group within a group footer. The following example student report includes control breaks for each student level.

Fig 4.19 Report layout

If producing different reports, try and ensure that there is consistency across them, i.e. ensuring dates and numeric data are displayed in the same formats and that report headers and page footers are in the same page position. As more systems access the Internet, using the web, email, mobile devices and social networking platforms, the UI requirements for these need to considered along with relevant design standards e.g. (United States government, n.d.). Newer technologies are emerging such as virtual reality and in the future these may be utilised to allow users to interact with systems in a more immersive way.

> Confidentiality needs to be considered when designing the UI. It is important to ensure that users only have access to the data they are permitted to work with. Likewise, care should be taken to ensure that any printed reports containing confidential or sensitive information are adequately controlled and managed.

Exercise 2

a) Produce a story board screen design suitable for entering essential data from the Invoice document shown in Exercise 1 above. Remember to include relevant control buttons.

b) Why would the following code not be suitable for accurately identifying a university student? 1234CMPY1. It consists of 4 unique digits followed by 3-character course code and Y1, Y2, or Y3 indicating the year of study.

c) What report type would be suitable for a user wanting to identify the best-performing student on a course?

Exercise 2 feedback

a)

b) Whilst the code appears to be informative, indicating the course and year, as the student moves through the years the code would be misleading and if it was changed it would cease to act as a permanent unique identifier. A similar problem would arise if the student changed courses.

c) An exception report – the report would only be showing the details for a single student.

4.7 SYSTEM ARCHITECTURE

Once the system has been designed from a software requirements perspective the system architecture needs to be specified. This is the technical infrastructure needed to support the system. It is particularly relevant if the system is to be implemented and maintained by the organisation using the system. If the system is to be outsourced, e.g. based in the cloud or supported by a software service provider, the infrastructure will in most cases be pre-defined by the suppliers. Cloud computing relies on using shared remote servers hosted on the Internet and this provides for greater flexibility, scalability and a different approach to financing without the need to implement and maintain the underlying infrastructure.

A number of factors can impact on the architecture chosen for a system, including existing corporate infrastructure, scalability to allow for growth in data volumes and number of users, any legacy (old existing) system requirements and any web integration needs. System security requirements may also have an impact on the infrastructure design. Initial costs and total cost of ownership will also need to be considered against the technical needs.

Traditionally, early commercial computing systems (1960s–1970s) were supported by mainframe (or mini) computers, usually based in and maintained by an organisation's central IT/computing department. These computers hosted all the organisation's systems and data and users interacted with them via dumb (no real processing capability) terminals that were connected directly to the mainframe. Mainframes are still used by some organisations which need to process large volumes of data at one location, e.g. banks. Mainframes were initially used with **batch processing** systems which involved entering large volumes of data and then processing the batches of data, often overnight, to update the main system files. Whilst this approach to data processing is suitable for less time-critical applications, it has been superseded in many cases by **online transaction processing** systems which process the entered data immediately in order to ensure that all data files are kept current.

The arrival of business personal computers (PCs) in the 1980s allowed users to host and run their own software and keep their own data, e.g. word processors and spreadsheet applications. Eventually these stand-alone PCs were connected to networks which allowed them to exchange data. Although stand-alone PCs provided some benefits they created issues relating to security and data consistency as different users may have used different versions of the same database.

Eventually **local area networks (LANs)** appeared. These allow stand-alone PCs (referred to as **clients**) to be connected to computers acting as **servers** which hold programs and data. These LANs also allow printers, document scanners and other specialist devices to be connected for sharing. **Wide area networks (WANs)** also exist, which allow the connection of clients or LANs over very large distances, e.g. in different countries. Systems using networks are often referred to as **distributed systems** and in most cases when a user accesses data via a network connection they are unaware of its underlying architecture.

There are a number of network configurations, although they are all regarded as client-server designs, and expert network architects are often used to specify and maintain complex networks. In a client-server configuration, the client sends a request for data over the network to the server. The server then processes the request and extracts the required data which is then sent back via the network to the client. Clients are often referred to as **fat** or **thin.** A fat (or thick) client is one which handles most of the application processing and is often a PC or laptop computer. A thin client does very little processing and instead relies on the server carrying out the processing, which usually results in better performance and a lower hardware cost. A thin client maybe a simple dumb terminal consisting of a screen, keyboard and mouse with no or limited processing capability.

Client-server designs are generally regarded as **two-tier** or **three-tier**. The clients in both designs handle the user interface. In a two-tier design the application processing is shared between the server and client, with all the data being stored on the server. In a three-tier configuration an application server sits between the client and the data server. This middle layer application server handles the client requests and breaks them down into data access commands that are handled by the data server. Some more complicated designs have multiple layers; these are referred to as **n-tier**. In multi-tier networks specialist **middleware** software allows for data transfer between the different levels, including enterprise applications and web services.

The choice of network configuration, i.e. data storage location and where the processing takes place, can have a significant impact on system performance and is often a balance between centralisation and decentralisation.

4.8   NETWORK TOPOLOGY

Network topology defines the configuration of a network, in particular the **logical** – the way the network components interact, and the **physical** – the actual components and connections. The **Open Systems Interconnection model (OSI)** defines the seven layers that perform functions within a network and how they interact with one another in order to allow data to be transferred from one device on a network to another. The seven layers (lowest to highest) are physical, data, network and transport which form the transport set, and the application set which consists of session, presentation and application.

There are three main network configurations – **Bus, ring** and **star**.

Bus topology

In the older, simpler topology bus network, all devices (clients and servers) are connected to a central bus (cable) which carries all the data between the devices. The advantage of this topology is that it requires less cabling than other topologies and allows for devices to be added or removed from the network at any time without causing disruption to other devices. Failure of a device on the network will not affect the other devices directly. However, the major disadvantage is that if the central bus fails, the whole network will fail to operate, also, as more users are added to the bus the performance declines.
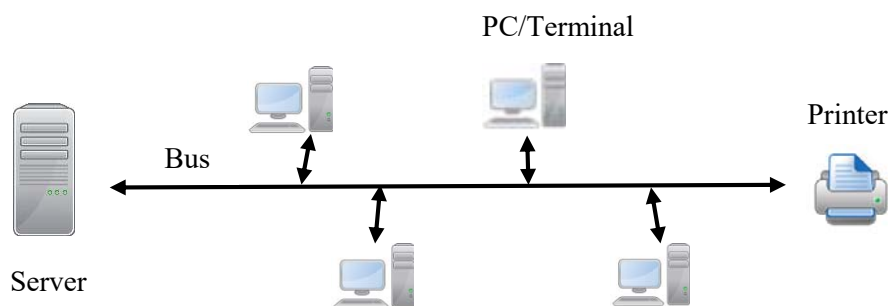
Fig 4.20 Bus network

Ring topology

Ring networks, although somewhat dated, are still used in some installations, mainly mainframe environments. Each **node** (device) on the network connects to two other devices and forms a continuous loop which allows data to transfer around the network. A disadvantage with a unidirectional ring is that if a node fails this can prevent the other nodes on the network from operating. Dual ring (counter rotating) networks can be configured to avoid this problem. A ring network can perform better than a bus network under heavy loads.
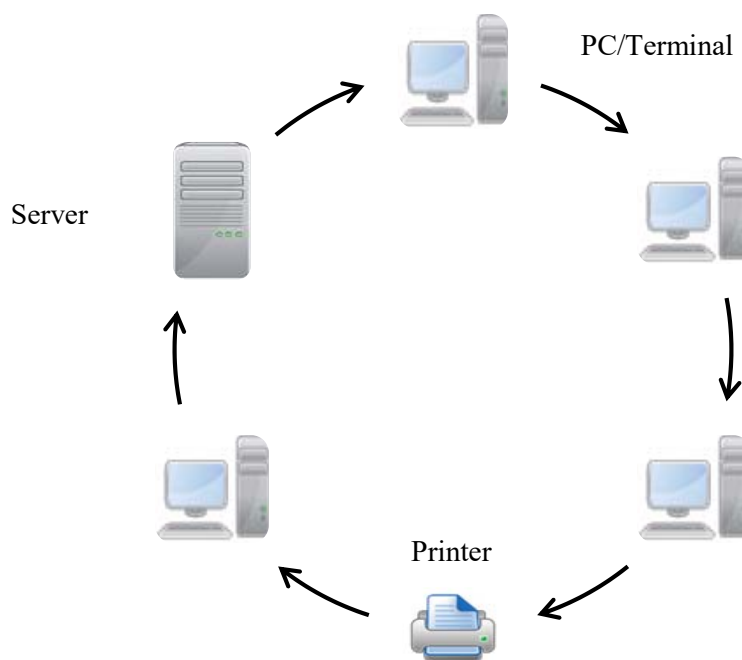
Fig 4.21 Ring network

Star topology

Star networks are the most commonly used topology. A simple star network contains a **switch**. This central node device connects to all the other nodes (devices) on the network and allows the network data to be routed through it. Each node is connected directly to the switch, which sends data to only the required devices. The advantage of the star network is that if one node device fails it does not affect the other nodes; however, if the switch fails, no computers can use the network, though this problem can be avoided by utilising a backup switch.
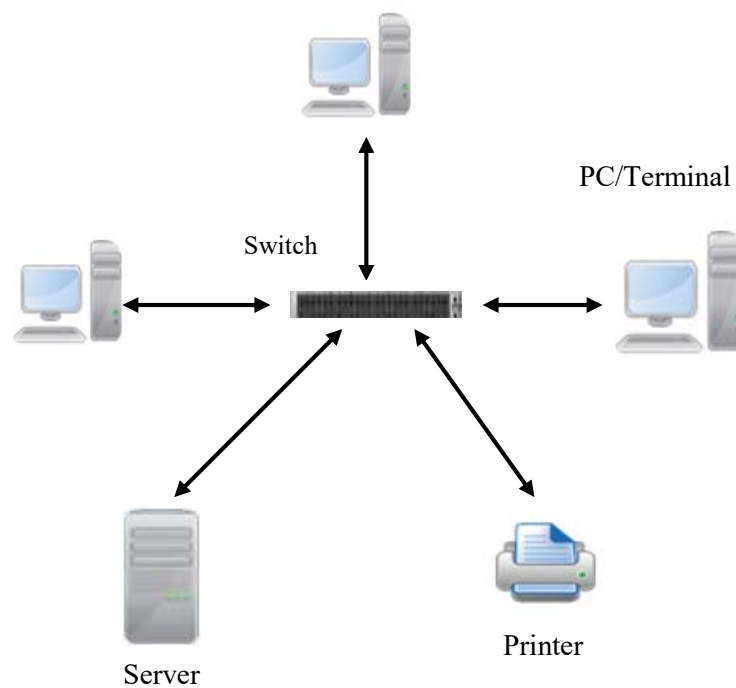
PC/Terminal

Switch

Server

Printer

Fig 4.22 Star network

Networks are commonly linked to other networks or the Internet by means of a connection called a **gateway** (proxy server) using a device called a **router.** A router chooses the best path for sending the packets of data. Protocols such as Internet Control Message Protocol (ICMP) are used to establish communications with other devices and choose the route between hosts.

With the growth of wireless networking technologies, many organisations are also using wireless local area networks (WLANs) as this saves on cabling costs and provides flexibility in terms of connected device numbers and location. However, there can be issues regarding security and electrical interference. There are a number of wireless networking standards e.g. Institute of Electrical and Electronics Engineers (IEEE) 802.11n, and these are constantly evolving to support newer technologies.

Exercise 3

a) What are the benefits of a wireless LAN?
b) Which is currently the most widely used network topology?
c) What device is used to provide a network gateway to the Internet?

Exercise 3 feedback

a) Relatively cheap to install and provides more flexibility in terms of adding additional users and their working location, i.e. office.
b) The star topology.
c) A router.

4.9   DESIGN DOCUMENTATION

Using a structured design approach when the software and hardware designs have been completed will generate a **system design document** which specifies the detailed design. The format of such a document varies between organisations but should include a management introduction, system architecture (hardware and software), design diagrams including the database design, human-computer interface (e.g. screen layouts), inputs (e.g. data entry screens) and outputs (e.g. reports, screen outputs). An example template can be found here:-

http://doit.maryland.gov/SDLC/Documents/sys_design_doc.doc (Maryland State Government)

Although much of the content, such as screen layouts, will have been reviewed and approved by system users during the design process, other stakeholders such as IT staff may need to consider the design and approve it. Similarly, management will need to be satisfied with the overall design and implementation implications relating to costs and staffing. The analyst will normally distribute the document to the stakeholders followed by a presentation in order to explain anything that is not clear and to answer any questions. The outcome will be one of the following – management approval to proceed, a request to undertake additional design work, or, in some cases, to cease work on the project. If approval is given the systems development phase can begin.

4.10 SUMMARY

Decisions need to be made as to how the proposed system is to be implemented. This may include outsourcing the development or procuring an existing software solution.

If the system is to be developed, the data requirements need to be modelled and the user interface designed. Most information systems use a database to hold their data so entity modelling, which is a top-down approach, can be used to produce a logical data model. A physical model is then derived which is used to produce a set of table designs. These are checked by means of the bottom-up approach of normalisation which ensures that a set of correctly-structured table designs is produced.

UI design is crucial to the success of a system and care should be taken to ensure that users are consulted in order to avoid costly changes at later stages in the SDLC.

The system architecture must be designed to take account of the system requirements and any organisational technical constraints.

The work of this phase is recorded in the system design document and is approved by all relevant stakeholders before the system development phase begins.

Further reading

(Gould, 2015)

(Shneiderman & Plaisant, 2010)

5   SYSTEMS IMPLEMENTATION

On completion of this chapter you should be able to:

- understand the software development process
- recognise software quality processes
- apply software design methods
- understand testing approaches
- recognise training needs
- understand changeover approaches.

The systems implementation phase of the SDLC includes developing the system from the system specification and testing it before it is implemented for the system users. As developing software **(application development)** is a costly process, it is important that quality control methods are used to ensure that the system produced satisfies the users' requirements. In order to achieve this, **software engineering** approaches are applied.

Software engineering focuses on developing software using well-defined design processes, documentation and testing regimes. The software engineering institute (SEI) (Software Engineering Institute, n.d.) developed a set of quality standards referred to as the **Capability Maturity Model (CMM)**® aimed at improving software development processes and therefore software quality. These standards classify an organisation's development processes using 5 levels of maturity and have been applied in many organisations in order to improve. A newer SEI technical report – **CMMI® (Capability Maturity Model Integration) for development** (CMMI Institute, 2010) outlines collections of best practices to help organisations to improve their processes. The five maturity levels are shown in order of increasing maturity as follows:-

Level 1 – Initial – the least mature, for weakly defined or ad-hoc processes.

Level 2 – Managed – processes are planned and executed.

Level 3 – Defined – processes are well defined and understood.

Level 4 – Quantitatively Managed – organisation has measured and controlled processes.

Level 5 – Optimising – Continuous process improvement.

The International Standards Organisation (ISO) has also produced a set of standards for software and systems engineering to help organisations confirm the quality of software systems (International Standards Organisation).

5.1 SOFTWARE DESIGN

In a traditional structured (or an O-O) approach to system development the next step will be to convert the design into a set of software modules/programs. These modules will be coded and tested to ensure they meet the specification, and at a later stage they will be amalgamated with other tested modules to form the complete system which will, in turn, have to be tested to ensure that all the modules interact correctly.

Although the bulk of the application development is undertaken by software developers the analyst may play a role in the software design and will also be involved in some aspects of the testing. The information needed to help design the software will have been collected and documented in the earlier stages of the SDLC and much of this will be held within the CASE tool repository.

Agile methods of software development do not rely on trying to design all of the software from the outset, as the emphasis is on iterative development which involves developing some code for a specific function, testing this and then gaining feedback from the users promptly. This process is repeated until the software is deemed acceptable.

Where a structured approach to software development is used a top-down process of decomposing the software functionality into modules is adopted, similar to that of a DFD. This modular approach involves producing a set of **structure charts** which show the software modules and how they are linked to one another.

The following structure charts illustrate modules and how they interact with one another. Each module is shown as a rectangle. Each rectangle has an identifying number. The higher level modules are referred to as control modules as they control (utilise) the subordinate modules that are linked to them. In the following example the control module is labelled 1. It controls sub-ordinate modules 1.2 and 1.3.

It also uses a library module, illustrated with two vertical bars. A library module contains re-usable code that may be utilised from any part of the chart. Arrows emanate from the control module to the subordinate modules. Where data passes between modules this is referred to as a **data couple** and is shown by means of a small arrow with an open circle pointing in the direction the data travels. In some situations, data, in the form of a flag which indicates a particular status, is sent from one module to another, e.g. 'student enrolled'. This is referred to as a **control couple** and is represented by a small arrow with a solid circle.
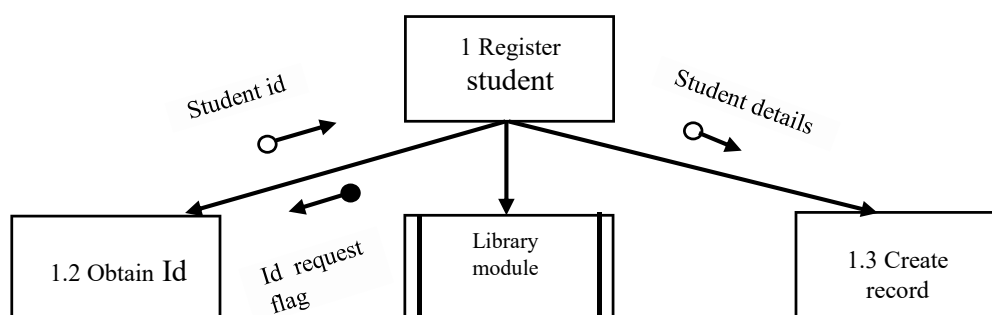
Fig 5.1 Example structure chart

In some cases, a control module determines which subordinate module is called, based on a specific condition. This **condition** is shown as a diamond on the end of the arrowed line. For example, a control module in a university admissions system may check a student's status and then decide to call either an undergraduate admissions module or a post graduate module.
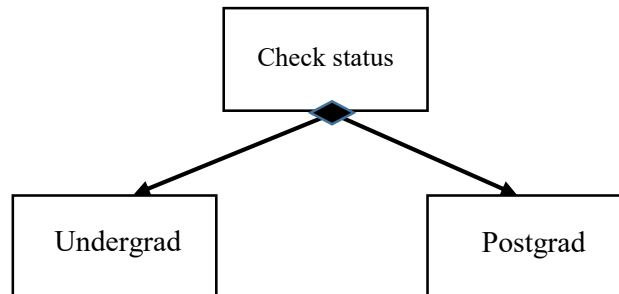
Fig 5.2 Structure chart Control module with a condition

In some situations, a module or set of modules may be repeated. This is shown by means of a **loop** around the arrows linking to the repeated subordinate modules e.g. a create invoice module will add an item then calculate the total for the item; this will be repeated for each invoice item and finally an overall invoice total will be calculated.
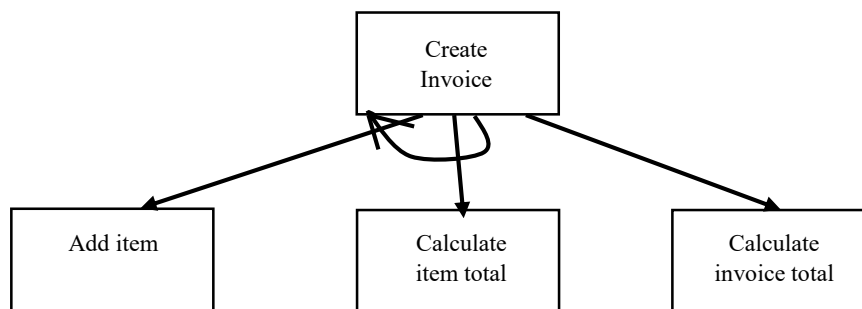
Fig 5.3 Loop showing 2 modules repeated

When designing modules, consideration should be given to producing reusable (library) modules to help reduce the overall amount of code. An example of this approach is to design a generic validation module which may be used by different parts of the system for checking that input information such as telephone numbers or dates of birth are correctly formatted.

When designing the structure chart it is desirable to try and produce modules that are highly **cohesive** but loosely **coupled**. **Cohesion** relates to the amount of processing a module undertakes; a highly cohesive module undertakes a single function. The benefit of this is that it will be a simpler module to code and test and it is also more likely to be reusable, e.g. a module that just checks a student_id as opposed to one that checks all of a student's details. Larger modules with more functionality should be considered for decomposition into smaller more cohesive ones. **Coupling** relates to the levels of interdependence between modules. Loosely coupled modules are independent, which makes it easier to modify them. In tightly coupled modules there is more dependence on shared data variables or control information.

The logical DFDs produced earlier provide a starting point for the structure chart hierarchy, with higher level processes usually translated into control modules which control lower level modules derived from lower level processes. The processes are decomposed until the primitive processes have been defined as modules. **Note.** Just as DFDs have multiple diagram levels, a structure chart may also show modules at a number of sub-levels. For example, in Fig 5.3 above the Calculate item total module may be linked to a lower level module which calculates the tax to be paid for the item.

The data items that are passed between modules can be identified from the DFD data flows and shown as data couples. Any loops and conditions should be added along with required control couples. The charts may need to be redrawn a number of times following input from software developers etc.

> **Try and organise the structure chart so as to have high cohesion and loose coupling.**

Where object-oriented development is being undertaken the object model, consisting of class diagrams and object relationship diagrams, (see Chapter 3) is utilised, as the class diagrams show the data (attributes) and the processing (methods) required.

The object methods are converted into modules, and messages are chosen to trigger them. The analyst or developer will consider sequence and state diagrams to help identify the messages used to trigger the modules. O-O software is often referred to as event-driven, as each message results in an action process being performed. Just as with structured development, cohesion and coupling needs to be considered; classes should be independent from other classes (loosely coupled). Similarly, an object's methods should be loosely coupled with other methods but be cohesive, i.e. carry out closely-related activities. A key aim is to produce code that is easy to understand and maintain.

Agile methods aim to speed up development by using alternatives to traditional project management, e.g. scrum development which involves using self-organising cross functional teams working on a small sub set of system requirements identified by a system user, often referred to as the customer or product owner. Each of these concise requirements is referred to as a **user story,** e.g. "A lecturer wants to be able to identify a student whose attendance falls below the expected norm". Each requirement is developed collaboratively using a series of rapid iterations called sprints (usually two weeks in length.) These effectively incorporate all phases of the traditional SDLC in order to produce usable software by gaining user feedback after each iteration so that improvements can be made. The user stories are normally prioritised by importance and the developers will assign a score that represents the level of complexity of the task, which is used to plan and allocate development resources. A plan for releasing the stories is drawn up and planning meetings allocate user story tasks to team members. The plan is modified as and when new requirements come to light. After each iteration the prototype is checked by the user and the cycle is repeated until all the user stories have been completed and accepted.

Agile development environments often use **pair programming** techniques referred to as **Extreme Programming (XP)**. This 'inspect-and-adapt' approach aims to improve software quality. It involves two programmers working on the same task together. One will do the coding (programming) and the other will observe and check the code. This method involves regular code reviews and discussions to ensure that the output is acceptable. The testing involves drawing up a test plan before the code is developed, and often automated testing is used with the aim of trying to break the code. There are regular meetings with the customer where feedback is obtained and any new requirements can be identified and incorporated.

Although Agile development is gaining in popularity there are some potential disadvantages, mainly in that the adaptive approach can allow for requirements creep, with time schedules becoming harder to predict as new requirements are added. The choice of development approach to be used will depend on a range of factors including type of system, time available and methodology expertise etc.

5.2 SOFTWARE DEVELOPMENT AND TESTING

The coding phase involves taking the design and producing the code in whatever programming language has been chosen, using a suitable **integrated development environment (IDE)** and any other required software tools such as report generators. The choice of language and development environment may be determined by the development organisation's standards and whether the system is to be web based. **Note**. Some program code may be automatically generated using CASE tools, e.g. database tables.

Software needs to be tested to ensure that it satisfies the system specification requirements and is reliable. **Test plans** are normally drawn up which indicate the nature and frequency of testing to be carried out. These cover **unit testing, integration testing, system testing and user acceptance testing**. In some cases, the analyst may be involved in setting the test scenarios and checking the results.

Typically, a programmer will test the individual modules they have written (unit testing). They will be looking initially to remove **syntax errors** caused by code not being written to conform to the language specification and structure, e.g. command word spelling mistakes. **Logic errors** also need to be eliminated. These are caused when the code executes but the desired results are not as expected. In the case of Agile development other programmers are involved in code reviews called **structured walkthroughs** to try and spot errors. Test data is prepared to test all possible situations, including the use of invalid data values. In order to test a module that will interact with others that are not yet completed, a **stub test** can be set up whereby a simulated value representing the other module's output is used.

Following successful unit testing, integration testing can take place, in which two or more related modules or programs are tested together to ensure that they interact correctly, i.e. handle data that is transferred between them. As with other types of testing, a range of data sets should be used, both normal and invalid, to thoroughly test the software.

When integration testing is complete, system testing can take place. This is when the whole system, comprising all the modules/programs, is tested. This type of testing ensures that the system meets the system specification and performance expectations, i.e. it can handle the predicted volumes of data without unacceptable performance degradation issues. When the developers and analysts are satisfied with the system test, users will be invited to take part in user acceptance testing to ensure that they, too, are satisfied that the system meets their requirements and expectations. In order to carry out this testing it may first be necessary to undertake user training for users, their managers and IT support staff. **Note**. This type of testing will need to be undertaken for both systems developed in-house and those that have been outsourced.

Whilst no system can be fully tested, the more thorough the testing is, the lower the risk of serious problems when the system is released for live use. Often a balance needs to be struck between the amount of time set aside for testing and the completeness of the system, and in some situations a system may be released with some functionality to be added at a later date.

5.3 DOCUMENTATION AND TRAINING

Before the system can be handed over for release, adequate documentation needs to be prepared. This involves ensuring that **program documentation** has been produced by the developers. This documentation is important as it will aid the developers in the future if bugs (software defects) are found or changes need to be made to the functionality. In addition to program documentation, **system documentation** is required. This describes how the system functions and will include material that will assist analysts and developers who need to support or enhance the system. Much of this documentation will have been created in the analysis and design phases of the SDLC and includes such items as screen layouts, data file definitions, models etc. If changes are requested and made to the system, they should be included in the documentation.

Depending on the nature of the system, **operations documentation** may also be needed. This is usually necessary when IT staff who are operating the system need to know when various batch reports need to be run or backups taken, along with any relevant security checks. Finally, **user documentation** is provided to ensure that the system users have enough clear information to enable them to make use of the system. This often includes a **user manual** which tends to be available on-line from within the system, rather than printed. In addition to the user manual, context-sensitive help may be made available from within the system, describing the available options to a user trying to use a particular screen or function. These materials are usually produced by the analyst, although in some organisations this may be carried out by members of the technical support team. Adequate time should be allocated to ensure the documentation is prepared and checked before the system is released for use.

Although in an ideal situation the system documentation should be sufficient to enable a user to make use of the system, most information systems require user training as well. Before a system is released the analysts will draw up a training plan to ensure that staff receive training in those aspects of the system they will be using. The amount and depth of training will vary depending on the level of system use to be undertaken. The training may be delivered by analysts, members of the development team or IT support staff. However, in cases where the system has been acquired as a package, or specialist hardware has been utilised, vendor training may be essential.

There are many ways in which training can be delivered and some or all may be used depending on the organisation's needs. As well as face-to-face training for individuals or groups, digital training resources may be made available utilising web or multimedia technologies to produce videos or online training simulators. In some situations, when groups of users have been trained they may go on to train other users.

5.4   SYSTEM CHANGEOVER

When a system is replacing an existing system a decision will need to be made as to how the changeover will take place. This is important, as usually data from the existing system will be needed within the new system. In some cases, it is relatively easy to export data from existing databases and import it into the new system's database using standard conversion formats. However, in more complex cases special conversion programs are developed to convert the data into the correct format in order to transfer it into the new system.

There are four main approaches for converting from the old system to the new one

- direct changeover
- parallel operation
- phased changeover
- distributed changeover

Direct changeover is where the old system ceases to be used as soon as the new system is implemented. This is usually the least costly approach but the riskiest, since if serious problems are encountered after the system is put into use it will not be possible to revert back to using the old system due to data incompatibilities. Also, direct changeover does not allow for checking outputs with the old system to ensure they are valid.

Parallel operation keeps the existing system in use alongside use of the new system. This allows for easier fall-back if the new system encounters serious problems but creates a potentially costly overhead of having to enter data into two systems. This approach allows for checking outputs to ensure they match with the old system but is not suitable if the two systems are not compatible.

Phased changeover allows the system to be implemented in stages. This means that the risks are reduced as only one part of the system will be affected if there are serious problems. This approach is less costly than the parallel operation and can be adopted if the new system can be modularised in such a way as to work with the existing system.

Distributed changeover is one in which the whole system is piloted by a part of the organisation e.g. by a branch of a company. The old system will still be in use by the organisation so if there are problems only a small number of users will be affected and fall-back is still possible.

Each of these methods has risks and cost implications and it will be up to the analyst, in consultation with other stakeholders, to decide on the best approach.

It is important that the system implementation includes physical, logical and behavioural security measures.

Physical security relates to hardware and software and involves ensuring this is kept in a secure environment with appropriate access controls in place to prevent unauthorised persons gaining access. Where networks are utilised, adequate firewalls need to be in place to prevent unwanted access. Further security measures may need to be implemented, particularly where systems utilise the Internet and may link with other third party systems such as on-line payment services.

Logical security refers to software controls within the system that prevent unauthorised users from accessing functions/data that they should not have access to. This involves using passwords and ensuring encryption techniques are utilised.

Behavioural security revolves around users of the system being made aware of safe operating and security procedures to prevent the system's security and confidentiality being compromised. This involves ensuring rules are implemented regarding the safe use of passwords and ensuring that sensitive system information such as printed reports are kept securely and destroyed when no longer needed.

The final stage of the implementation phase is to carry out a system review and produce an evaluation report for management. This takes place after the system has been implemented and is used to assess how successful it is in terms of meeting the user's specification and expectations and whether the expected benefits have been achieved. A further purpose is to reflect on the system and suggest any possible future system enhancements. The report will also highlight any valuable lessons that may have been learned relating to the project, including any technical issues, predicted and actual costs and planned and actual project duration, as these may be of benefit to other members of the IT development team for use in planning future system development projects.

5.5   SUMMARY

The systems implementation phase covers the application development or system procurement, testing, documentation, training, implementation and evaluation. Software development involves converting the system design into a set of suitably structured modules and programs. Testing plays an important role in ensuring that the system meets the specification and that it performs to an acceptable standard so as to ensure a quality solution is implemented. Documentation and training are also required so that the users can make effective use of the system. In order to implement the new system, a suitable changeover strategy needs to be adopted to minimise risk during the transition period. Finally, following implementation, an evaluation review should be carried out, resulting in a report which indicates any future system enhancements and highlights any technical or cost-related issues that may be relevant when undertaking future projects.

Further reading:
(James, n.d.)

6   SYSTEMS MAINTENANCE

On completion of this chapter you should be able to:

- describe system support activities
- understand system maintenance
- be aware of security issues
- understand backup and recovery procedures.

Systems maintenance is the last phase of the SDLC, and includes supporting the system so that it operates reliably and in a secure environment. Once a system has been released for use it is usually supported by the IT support team, although in the case of a system that has been outsourced this support may be provided directly from the vendor. The IT support team may include administrators, technicians, maintenance programmers (software developers who tend to existing systems) and systems analysts, some of whom will have a thorough knowledge of the system and so be able to identify and solve problems speedily.

6.1   USER SUPPORT AND TRAINING

A user's first point of contact is usually with the IT help (or service) desk. The help desk will have staff who are sufficiently trained to assist with most technical or operational queries. If the query cannot be solved by the front line staff, the issue may be escalated to more specialised staff or members of the system development team such as the systems analyst. Requests and problems are normally logged by the help desk and are reviewed periodically to see if there are recurring issues that need addressing, perhaps by training or system enhancements. The IT help desk is normally also responsible for system training, as this is necessary for new staff members and for existing staff when changes or upgrades are made to the system. In addition to dealing with any software related queries the IT support team will also be responsible for assisting with any hardware or technical problems i.e. replacing a faulty PC or installing a new version of the system software.

6.2   SOFTWARE MAINTENANCE

Over the lifetime of a system various types of maintenance will be needed to ensure the system continues to satisfy the users' needs. There are four main types of maintenance:-

Corrective maintenance relates to dealing with faults in the hardware or software. Software faults are often referred to as "**bugs**"; these cause unexpected or incorrect results and in serious cases cause the system to stop functioning. These faults will often need the software programs to be **debugged** (checked and the faulty code corrected). Also, hardware problems such as faulty PCs or network components can arise and will need to be addressed. The IT team will need to assess the seriousness of the faults in order to decide what action needs to be taken and by when.

Adaptive maintenance addresses changes that need to be made to the system to deal with new demands. These may be changes imposed in order to satisfy the requirements of an outside agency, such as the government introducing changes to the tax regime that involves altering tax calculations.

Perfective maintenance relates to changes made to the system in order to improve its functionality or performance, for example, adding a new report or upgrading some system hardware to increase response times.

Preventative maintenance is carried out in order to prevent future problems. This may involve making changes to cope with increasing volumes of data or transaction levels that may result in performance degradation issues if not addressed.

The costs of supporting a system over its lifetime can be significant, therefore support needs to be managed carefully, including monitoring and prioritising maintenance tasks. Essential tasks are those needed to keep the system functioning, whereas desirable ones may include adding new features.

There are usually a number of **maintenance releases** of the system incorporating changes. Care needs to be taken to ensure that there is adequate **version control** of the releases. This involves maintaining accurate documentation regarding the changes and ensuring that backup copies of the different software versions are maintained in case it proves necessary to revert to an earlier version following unexpected problems with the new version. Software packages such as **git** https://git-scm.com/ are available for this purpose. **Note**. Support costs tend to be high in the period following system implementation until the system is bedded in and then they rise again towards the end of its life as more effort is needed to keep it functioning effectively.

6.3 SYSTEM PERFORMANCE

Whilst the system is in use its performance and reliability are monitored and managed by the **system administrator** to ensure that it is coping adequately with the demands placed on it. Fault management procedures are also needed to manage problems such as system failures, user-caused problems, environmental problems such as power failures and even natural disasters such as flooding or earthquakes.

Response times need to be maintained to agreed levels otherwise users will become frustrated and key business deadlines may be missed. To avoid performance problems, **capacity planning** needs to be carried out. This includes monitoring and forecasting throughput levels, e.g. the number of and time taken to carry out transactions such as making a hotel room booking. This is necessary to allow for capacity changes such as increased workload levels or the addition of more users. Changes implemented to address these issues can include altering usage patterns of the system, by running certain programs at off peak times, upgrading hardware or communications links or altering the software. Specialist tools may be utilised to help monitor aspects of system performance such as network diagnostics packages.

6.4   SYSTEM SECURITY

As information systems are essential to organisations, it is crucial that they are available when required and that they continue working correctly even when events such as deliberate threats, accidents or disasters occur. Establishing a secure system is a complex business and covers not just the hardware and software but also the way in which people interact with the system.

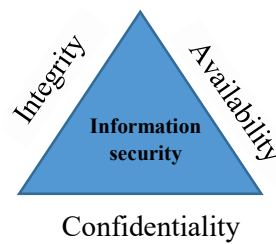The CIA triad is a model which shows the three main goals for information systems security.

Fig 6.1 CIA triad

Confidentiality is the protection of information from unauthorised access. This is particularly important when personal or sensitive information is to be managed. It can be maintained safely if suitable access controls are put in place to ensure users can only access permitted information.

Integrity is the keeping of accurate information. It is important to ensure that information cannot be changed or lost due to errors or improper access. Access controls can be put in place to avoid this.

Availability is making information available when and where needed. This means ensuring that the system can keep functioning so as to provide the required information. Security mechanisms need to be in place to ensure that the system components are secure and that the system can continue to operate under a range of conditions.

The above goals must be considered as part of the **system security policy**. In order to produce the policy, it will be necessary to undertake a risk assessment to identify likely threats and how vulnerable the system would be to them. Typical threats may include hardware or software failures, human errors, power failures, physical damage or theft and software attacks such as viruses or ransomware. Once risks have been identified, preventative actions can be taken to address them.

Physical attacks can be addressed by putting in place adequate physical access and security around hardware locations such as server rooms. Where computer centres are used these will normally have user access controls in place to prevent unauthorised persons gaining access to the centre, usually involving electronic door access systems. Where PCs, laptops and smaller IT devices are located in more open environments such as offices, they can be secured with physical locking mechanisms.

In order to ensure that the system hardware can function unimpeded by power cuts, backup generators or uninterruptable power supplies can be utilised. For highly critical systems a disaster recovery plan will be developed to allow an organisation to continue its operations in the event of a major disruption, which may include maintaining a duplicate system at a different geographic location which can be put in to operation promptly.

Networks also need to be secured, which involves ensuring data traffic is encrypted and, in the case of wireless networks, ensuring that security protocols such as **Wi-Fi Protected Access 2 (WPA2)** are implemented. In order to prevent unauthorised access to the network via the Internet a **firewall** needs to be in place. In addition to firewalls, **host and network intrusion detection systems (HIDS/NIDS)** can be used to identify possible threats from **hackers** or **crackers**.

Typical software controls include **application permissions** which restrict user access to only specified parts of the system, usually controlled by means of passwords assigned by the system administrator. **Access logs** are also used in some situations which log users' access to the system and can be used to highlight misuse. The integrity of the system data is vital and is aided by validation techniques within the software to ensure that only valid data is entered.

Files or database tables can have permissions placed on them which limit a user's access rights. **Read** allows a file to be read, **Write** allows the file to be written to, **Execute** allows a program file to be run. These rights may be assigned to user groups or to individual users.

Operational security revolves around having procedures in place that all staff adhere to in order to ensure that security is maintained. This covers the safekeeping of passwords and system outputs such as reports or file copies.

Finally, **backup procedures** need to be established to ensure that the data and software are kept safely to allow for recovery in the event of a service disruption. Recovery will involve reloading program or data files that may have been lost or corrupted. There are four main approaches to backups:-

Full backups where all system files are copied. This method takes the longest and uses the most storage but does allow for a simpler and quicker recovery.

Incremental backups only back up files that have changed or are new since the last backup was made. This type of backup is quicker than a full backup and uses less storage but will take longer to restore as more than one backup may have to be restored.

Differential backups are similar to incremental ones in that they backup files that are new or have changed since the last full backup. This is faster than a full backup and uses less storage. Restores can be completed using the last full backup and the last differential backup.

The above backup methods are usually performed monthly, weekly or daily as required. The backup files should be kept in a safe location away from the in-use versions so as to minimise the risk of them being lost in the event of a disaster.

Continuous or mirror backups involve mirroring or copying the system files continuously. This is the costliest approach as the hardware and network infrastructure must support it, however it does provide for fast restores to a point before a problem was detected.

6.5   SYSTEM TERMINATION

At some point the system will reach the end of its useful life and cease to be of use to the organisation. This may be because it no longer satisfies the organisation's operational requirements, because it has become too costly to maintain, or because new technologies may offer a better solution. In time a decision will be made to phase the system out and replace it. If a new one is required, then the SDLC will start over again. The systems analysts, users and other stakeholders will be consulted to help make the decision.

6.6   SUMMARY

The system maintenance phase operates from when the system is released for use and continues until the system is no longer in use. During this phase new users are trained and the system is maintained in order to deal with any errors or performance issues and also to add new requirements. System performance is monitored and changes made to address any issues. System security is vital, and procedures are put in in place to ensure that the system functions reliably in a safe and secure environment. This includes ensuring suitable backup procedures are in place to ensure that the system can be restored in the event of a significant disruption. Eventually, the system will cease to be suitable and a decision may be made to replace it, which will cause the SDLC to be restarted.

7 BIBLIOGRAPHY

Agile Models Distilled: Potential Artifacts for Agile Modeling. (n.d.). Retrieved from Agile Modeling: http://www.agilemodeling.com/artifacts/

agilemanifesto.org. (n.d.). *Manifesto for Agile Software Development.* Retrieved from agilemanifesto.org: http://agilemanifesto.org/

Aguanno, K. (n.d.). Retrieved from The AGILEPM: http://agilepm.com/freestuff

Ambler, S.W. (2009–12). *The Agile System Development Life Cycle.* Retrieved from Ambysoft: http://www.ambysoft.com/essays/agileLifecycle.html

Beck, K., et al. (2001). Retrieved from http://agilemanifesto.org/: http://agilemanifesto.org/principles.html

CMMI Institute. (2010, November). *http://cmmiinstitute.com/cmmi-models.* Retrieved from CMMI Institute: http://cmmiinstitute.com/system/files/models/CMMI_for_Development_v1.3.pdf

Connolly, T. &. (2015). *Database Systems A Practical Approach to design, Implementation and Management 6th Ed.* Pearson Education.

Gould, H. (2015). *Database Design and Implementation A practical introduction using Oracle SQL.* Bookboon.com.

Hay, D., & Lynott, M. (2008). *TDAN Newsletter.* Retrieved February 23, 2015, from http://www.tdan.com/view-special-features/8457

International Standards Organisation. (n.d.). *Standards catalogue.* Retrieved from International Standards Organisation: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45086

Introduction To OMG's Unified Modeling Language® (UML®). (n.d.). Retrieved from Object Management Group: http://www.omg.org/

James, M. (n.d.). *Scrum Training Series.* Retrieved from http://scrumtrainingseries.com/

Mark Dixon. (n.d.). Retrieved from QSEE TECHNOLOGIES:
http://www.leedsbeckett.ac.uk/qsee/

Martin, J. (1991). *Rapid Application Development*. Macmillan.

Maryland State Government. (n.d.). *System Design Document Template*. Retrieved from Maryland.gov: http://doit.maryland.gov/SDLC/Documents/sys_design_doc.doc

Net Present Value. (n.d.). Retrieved from Investopedia: http://www.investopedia.com/calculator/netpresentvalue.aspx

Object Management Group Business Process Model and Notation. (2014). Retrieved from OMG: http://www.omg.org/spec/BPMN/2.0.2/

Project Management Institute. (n.d.). Retrieved from Project Management Institute: http://www.pmi.org/default.aspx

Royce, W. (1970). Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON* (pp. 1–9). TRW.

Schwalbe, K. (2010). *Managing Information Technology Projects 6th Edition*. Cengage Learning.

Shneiderman, B. (2010). *The Eight Golden Rules of Interface Design*. Retrieved from university of Maryland: http://www.cs.umd.edu/~ben/goldenrules.html

Shneiderman, B., & Plaisant, C. (2010). *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Fifth Edition*. Reading, MA: Addison-Wesley Publ. Co.

Software Engineering Institute. (n.d.). *Process & Performance Improvement*. Retrieved from Software Engineering Institute Carnegie Mellon University: http://www.sei.cmu.edu/process/

United States government. (n.d.). *U.S. Web Design Standards*. Retrieved from https://playbook.cio.gov/designstandards/

What is PRINCE. (n.d.). Retrieved from PRINCE2.com: https://www.prince2.com/uk/what-is-prince2

Wiegers, K.E. (1999). Retrieved from McMaster University: www.cas.mcmaster.ca/~carette/SE3M04/2003/files/srs_template.doc

8   APPENDICES

8.1   APPENDIX A – COST BENEFIT ANALYSIS

In order to help decide whether it would be appropriate to proceed with a particular information system, a cost benefit analysis (CBA) is normally undertaken for each possible system proposal. This compares the anticipated costs with the expected benefits. There are a number of different CBA techniques and this appendix shows some of the popular ones: – **payback analysis, return on investment (ROI)** and **net present value (NPV)**.

Payback analysis

This determines the length of time it will take for a system to pay for itself. The time taken to recoup the costs is called the payback period. The analysis involves working out the development cost of the system and its annual operating costs. The annual benefit costs are also worked out. The payback period is found by comparing the accumulated total costs to the accumulated total benefits. In the example below the payback period is in year 4, as the cumulative benefits of 123500 exceed the cumulative costs of 116300. **Note**. Year 0 is the development year, although as the system was implemented within that year some benefits are accrued.

System 1				
Year	Costs	Cumulative costs	Benefits	Cumulative benefits
0	50000	50000	2500	2500
1	15000	65000	25500	28000
2	16000	81000	30000	58000
3	17300	98300	32000	90000
4	18000	116300	33500	123500
5	19500	135800	35000	158500

Fig 7.1 Payback analysis

Some organisations set a payback period and if a project does not show a payback within that time the project may not proceed, even though the benefits may exceed the costs over a longer period of time.

Return on investment analysis (ROI)

The ROI% can be used to measure profitability by comparing the return (total net benefits) from the system against the investment (the total costs) as follows:-

ROI = ((total benefits – total costs) / total costs) × 100

Using the data from Fig 7.1 above gives: –

((158500 – 135800) / 135800) × 100 = 16.7% ROI

Organisations often set a minimum ROI which may be based on the rate they could obtain by investing the money in other ways, such as an investment bank account. Note. This approach considers the overall rate of return over the lifetime of the system and does not consider annual rates of return, which could vary significantly. ROI can be used to compare different systems proposals. It is, however, worth noting that the timing of the costs and benefits is not considered. These shortcomings are addressed when carrying out a net present value analysis.

Net present value (NPV)

The NPV can be used to consider the profitability of a system and is the difference between the present value of cash income (benefits) and the present value of cash outgoings (costs, including initial development costs). The time value of money works on the basis that a unit of currency held now is worth more than the same unit of amount which would be received in a year's time, as money currently held can be invested and could be worth more in the future.

The present value of a future unit amount e.g. $ or £, is the amount of money invested at a specific interest rate today which grows to become the future unit value at a specified future point in time. This specified interest rate is referred to as the discount rate. An organisation will choose a discount rate that represents the expected rate of return from investing in a safe form of investment such as a savings account, rather than invested in a new system. Organisations normally expect to see a rate of return that is higher than the discount rate in order to take account of the increased risk of developing a new system rather than just investing the money. The NPV formula involves multiplying each of the incomes and outgoings by the relevant present value factor which is based on the year the incomes/outgoings will happen. Then all the time-adjusted incomes and outgoings are totalled. Finally, the NPV can be calculated by subtracting the total present value of the costs from the total present value of the benefits

$$NPV = \sum \{Net\ Period\ Cash\ Flow/(1+R)^T\} - Initial\ Investment$$

where R is the rate of return and T is the number of time periods.

For example, take a project with an initial cost of £20,000 over 3 years with a discount rate of 3.5%.

$NPV = \{£2,000/(1+.035)^1\} + \{£10,000/(1+.035)^2\} + \{£12,000/(1+.035)^3\} - £20,000$
$= £1932.37 + £9335.11 + £10823.31 - £20,000$
$= £2090.79$

To assist with the present value analysis, present value tables are available which show values at various rates over a number of years. There are also Internet calculators available (Net Present Value, n.d.)and spreadsheet software such as Microsoft® Excel can be used for the calculations.

If the NPV value is positive, the system would be feasible as it would return more money than the amount invested, however all development projects have risks and so organisations will still need to consider the return carefully. NPV analysis can be used to compare different system proposals.

8.2 APPENDIX B – NORMALISATION TEMPLATE

UNF	1NF	2NF	3NF	Relation / Table Name
1. List all the attributes below from a single document / table. 2. Identify the unique identifier / primary key. Show in bold or colour. May need an artificial key. 3. Identify any repeating attribute group(s). Show inside (….) or colour.	1. Place repeated attribute group(s) if any in a new relation. 2. Include the UNF unique identifier as a foreign key in the new relation. 3. Identify the additional attribute(s) in the new relation to form a compound key with the foreign key.	1. Remove any part key dependent attributes to a new relation. 2. Identify identifier for each new relation. 3. Include foreign key in the original relation.	1. Remove any non-key dependent attributes to a new relation(s). 2. Identify the unique identifier for the new relation(s). 3. Include a foreign key in the original relation.	Assign a suitable name for each relation/table.

8.3 APPENDIX C – PROJECT MANAGEMENT

In traditional systems development projects, the analyst or a project manager will be responsible for managing the project, including planning, scheduling, monitoring and reporting to relevant stakeholders. **Project management** focusses on managing the balance of costs, time and project scope, sometimes referred to as the **project triangle**, as a change to one of the three will affect the other two.

Project management involves identifying the tasks to be carried out and producing a **work breakdown structure (WBS)**. The WBS is used to produce a **Gantt chart** which shows the schedule of tasks on a horizontal bar chart. **Program Evaluation Review Technique (PERT)** and **Critical Path Method (CPM)** are two important techniques used to calculate the length of the project and show the **critical path** which is made up of the sequence of activities that add up to the longest overall duration. This determines the minimum time possible in which to complete the project.
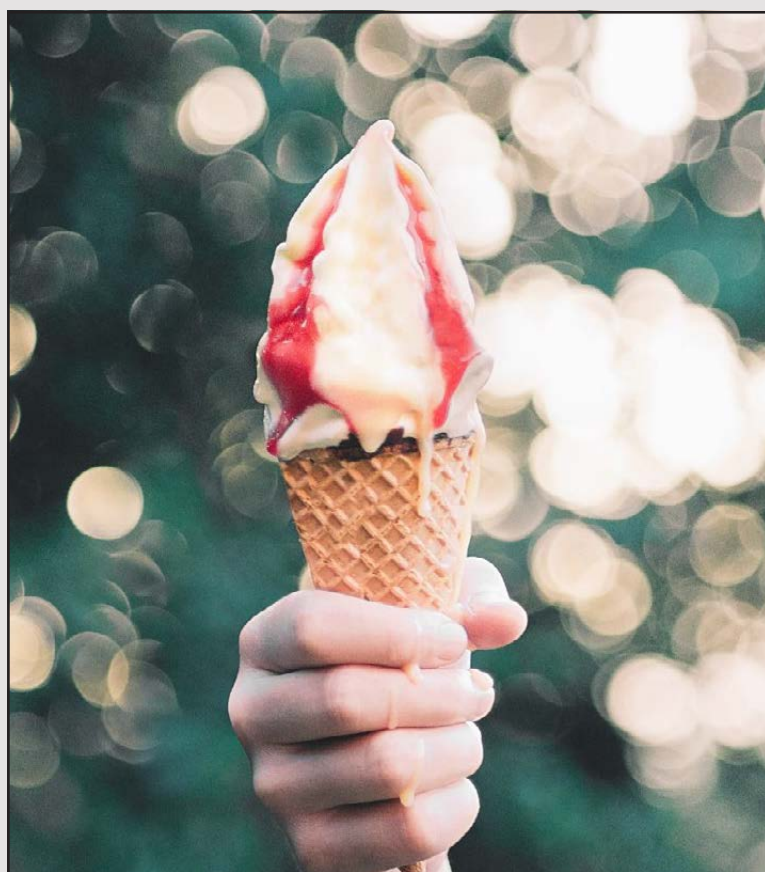
A work breakdown structure is formed by breaking down all the main **activities** in the SDLC into smaller **tasks** and assigning them a duration, costs and resources, e.g. staffing. In addition to the tasks there are also milestones, important events occurring at a set point. These are used to monitor progress and usually involve attention from stakeholders. A milestone may be used to signal the end of a deliverable such as "testing completed". Tasks may be dependent on other tasks e.g. one task cannot start until another has finished.

The following example shows the work breakdown structure for a system development project in the form of a Gantt chart using Microsoft project professional 2016. The main activities are shown in the Task name column and each of these higher level activities can be expanded to show their lower level tasks, as shown for the activity "Analysis/Software Requirements", which reveals 9 sub tasks. The horizontal bars to the right of the tasks indicate the task duration in days. The small arrow exiting a task bar indicates that when this preceding task is completed the task being pointed to can start. **Note**. In most projects there will be some tasks that can run in parallel e.g. testing, where different modules may be undergoing testing at the same time. Milestones are shown as black diamonds and indicate a project check point which usually has no duration.
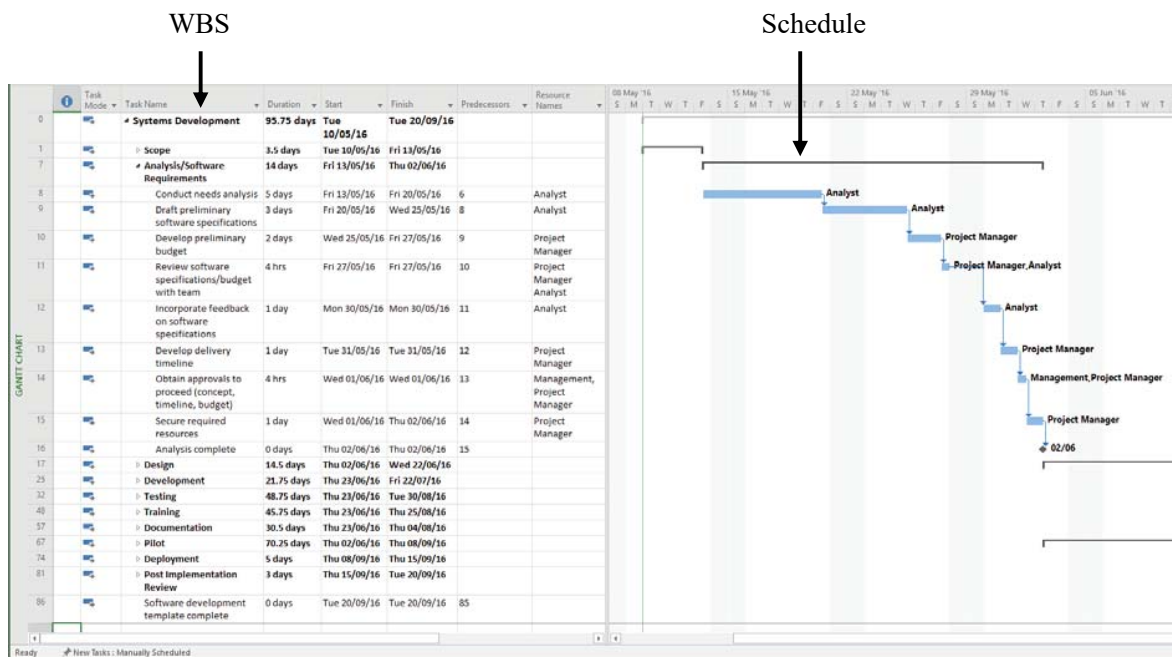

Fig 7.2 Example system development Gantt chart

WBS tasks may be arrived at either using a top-down approach of breaking down high level activities such as an SDLC phase, or using a bottom up approach which involves identifying all the tasks and then grouping them under higher level activities. Where a similar project has been undertaken it may be possible to utilise an existing WBS template. Accurate task duration estimation can be difficult although experienced analysts and project managers will find this easier.

When all the tasks have been entered the CPM, which is also called the critical path analysis, can be performed in order to identify the critical path for the project. This is the series of activities which determines the earliest time the project can be completed. It is in effect the longest path through the project with the least amount of **slack** or **float** – the amount of time that a task can be delayed without delaying a succeeding activity or the whole project.

The critical path can be calculated by using a **network diagram** which is derived from the WBS.

There are two types of network diagram, **activity-on-arrow (AOA)** or **arrow diagramming method (ADM)** and the **precedence diagramming method (PDM)** or **activity-on node (AON)**.
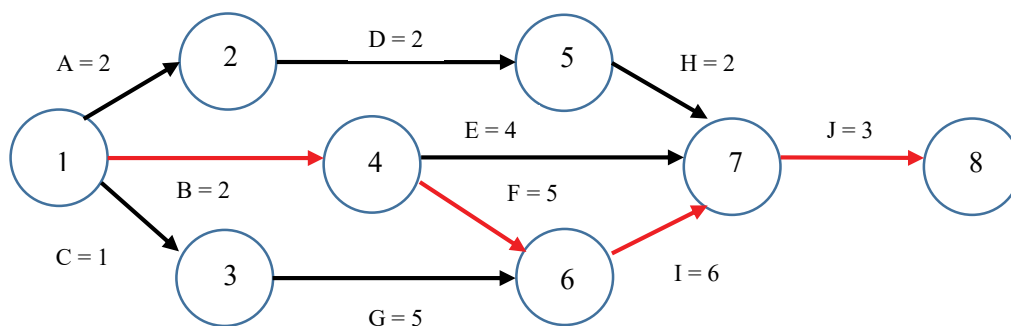

Fig 7.3 Activity-on-arrow (AOA) network diagram

The activities are labelled A-J and their durations are shown in days.

This gives the following paths:

1. A-D-H-J   2+2+2+3 = 9 days
2. B-E-J   2+4+3 = 9 days
3. B-F-I-J   2+5+6+3 = 16 days
4. C-G-I-J   1+5+6+3 = 15 days

Therefore, the critical path is path 3 which is 16 days – the longest path through the network.

The precedence diagram method shows the activities within boxes with the arrows linking them.

The example precedence diagramming method network diagram below, which was produced using Microsoft Project, shows the same activities as used in the network diagram above. Each activity is shown as a box which contains the task start date, finish date and duration. The activities highlighted in red are critical activities which cannot slip without affecting the overall project deadline.
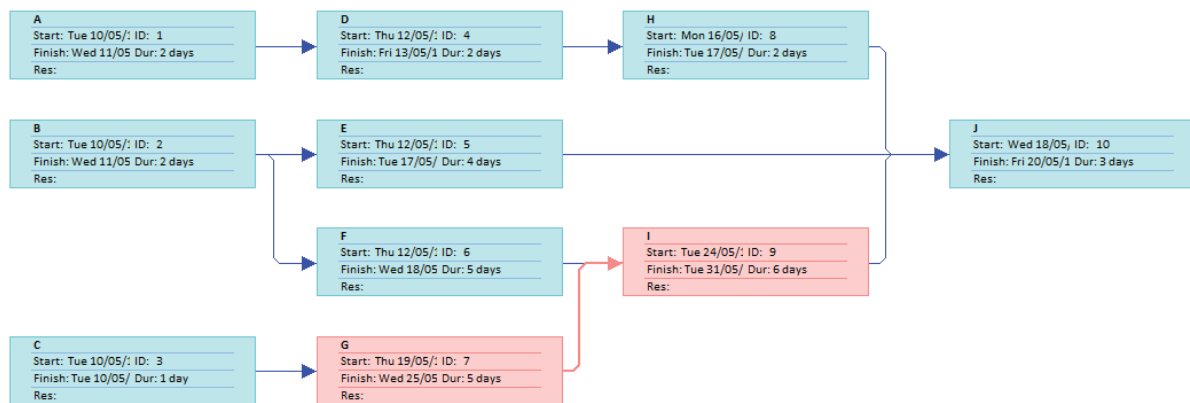
Fig 7.4 Precedence diagramming method network diagram (AON)

The schedule with the planned dates is referred to as the **baseline schedule** and the analyst or project manager would use the Gantt chart to track the project's progress. Actual dates of task completion would be recorded at regular intervals so as to help track the project and to enable remedial action to be taken to minimise problems if milestones are missed.

Note. Project management software tools include the facility to record a project plan and the progress made against it and also make it easy to produce reports for distribution to relevant stakeholders.

Agile project management

In an Agile development environment, the long cycle of the conventional waterfall model is broken down into small segments of the product which are specified, developed and tested in more manageable periods of two to four weeks. By receiving continuous feedback, problems can be dealt with much earlier. The product owner handles project goals and balances the schedule against the scope, prioritising product features. The scrum master helps the team to prioritise their tasks and the team members deal with the task assignment, progress reporting and product quality control.

Further reading
(Schwalbe, 2010)
(What is PRINCE, n.d.)
(Project Management Institute, n.d.)
(Aguanno, n.d.)