# Chapter 15

# IT Systems in Organisations


## Objectives

1. Explain the challenges associated with IT project development
2. Explain the steps of the System Development Life Cycle (SDLC)
3. Explain the methods used to manage IT projects
4. Construct PERT and Gantt charts to model a project's schedule
5. Construct ERD and DFD diagrams to model an IT system
6. Discuss appropriate project management techniques and decisions

# IT Systems in Organisations

As demonstrated throughout the ITGS course, organisations and individuals in all areas of society rely heavily on information technology. Yet designing, developing, installing, and changing over to new IT systems is still an area that causes great problems for many organisations, and often results in failure and wasted money. This chapter examines the steps involved in analysing, designing, developing, using, and supporting IT systems effectively in organisations.

## Types of Development

When an organisation decides to investigate a new IT project, it can choose from two fundamental types of software to meet its needs: off-the-shelf software and custom / bespoke software. **Off-the-shelf software**, as its name suggests, is software that is widely available for general purchase from software vendors. This might be general application software used by many types of organisations, such as office software suites, or it may be more specialised to a particular type of business (for example, school information management systems).

**Custom / bespoke software** is created specifically for a single organisation. Usually the organisation hires a software development company, but it may also have its own software developers in house. The developers analyse the organisation's needs and design software specifically to meet them.

**Figure 15-1** An Integrated Development Environment (IDE) offers programmers many tools to help them create software

Businesses might employ bespoke software if it is more tailored to their needs than off-the-shelf software packages. At the larger end of the scale, corporations like NASA or Boeing contract software developers to create their software (it is obviously very difficult to walk into a shop and buy a shuttle control program off-the-shelf!).

## Development Tools

Software developers use a variety of programs to create new software. A text editor is used to enter the program's **source code**, a compiler translates the source code into machine code, and debugging tools to help find and fix errors in the program they are creating. Often these tools are packaged together in an **Integrated Development Environment** (IDE). Many IDEs also offer tools to create graphical user interfaces for programs by dragging and dropping components on the screen.

## Legacy systems

A **legacy system** is a computer system that is no longer available for purchase or is no longer supported by the manufacturer. A legacy system might be just a few years old, or it could be decades old. Some legacy systems may operate on (and even require), certain very old hardware which is no longer available. Others may only run on older operating systems and not be compatible with modern versions. Sometimes the manufacturer of a legacy system no longer exists, and in other cases the manufacturer has dropped support in favour of more recent products. This usually means updates and security fixes will not be available for the system, which can be a significant problem for organisations.

Often organisations continue to use legacy systems because they are essential to their operation and there is no easily available replacement. Replacing a legacy system with a new one may be cost prohibitive, and with very old software, it may be extremely difficult to convert data from a legacy system to a new format. This might be made harder by a lack of knowledge of the system: the original designers of the system may have retired, documentation may be absent, or undocumented changes may have been made since the original installation.

If the system was written in a older programming language, it can be very difficult or expensive to find soft-

## Off the Shelf Software

| Advantages | Disadvantages |
|---|---|
| There are many more users, meaning there is usually more support available in terms of documentation, books, and user groups. | Off-the-shelf software is a 'one size fits all' solution: it might not do everything the organisation wants it to do, and it might include many features which they pay for but don't need or use. |
| Data and file compatibility with other systems (including customers' and partners' systems) may be greater because the software is widespread. | Because the software vendor has thousands and thousands of customers, a single organisation is not very important to them in the grand scheme of things. This means that if they want a change or a new feature, they are very unlikely to get it unless it is in great demand by many users. |
| The cost is usually lower. | |
| Installation may be easier. | |
| The software vendor will have a large 'knowledge base' of common errors and their solutions. | |
| Software has been extensively tested (both by the vendor and by thousands of users!). This reduces the number of serious bugs the software contains. | |

## Custom / Bespoke Software

| Advantages | Disadvantages |
|---|---|
| The software can be customised to the organisation's precise needs, including features and the user interface. | Selecting a competent software developer is difficult, especially for managers who may not have software development experience. |
| Changes or additional features can usually be added later on request. | The price is significantly higher than off-the-shelf software. |
| Having software developed to your precise needs can give a great competitive advantage. | Specifying precise requirements for projects, especially large projects, is difficult and error-prone. |
| | Development can take a long time. |
| | If the software developer does not provide the source code to the project, you are locked-in to using them for help and support. |
| | If the software developer's business fails, you are left completely without help, support, or upgrades (this is true for off-the-shelf software companies too, but bespoke developers are often to be smaller operations, and more vulnerable to market forces) |

ware developers who have the necessary skills to understand the program's source code—or the source code may just be unavailable.

Finally, if a system works, an organisation may not want to replace it. This is especially true if the only problem with the legacy system is its age (i.e. it performs all the desired functions). In these cases it is quite common to develop a new user interface which interacts with the legacy system 'behind the scenes' —such as a web based interface to replace a text-only command line interface.

Nevertheless, legacy systems cause problems for organisations. Older hardware and older operating systems require personnel with rare skills to support them, and these people are often more expensive to hire. A 2011

report suggested that the US government spent 46% of its annual $36 billion IT budget on maintaining legacy systems[1].

## Emulation and Virtual Machines

One solution to using legacy systems in a modern environment is to employ a **virtual machine (VM)**, which creates a virtual 'computer' running inside a window, like a regular application program. This virtual computer is independent of the host computer and can be paused, shut down, restarted, and modified at will. It runs its own operating system and software, independent of the host computer. Virtual machine software typically allows configuration of the virtual computer's hardware, including the amount of RAM and the size of the hard disk. Often a single large file is used to represent the 'hard disk' of the virtual computer. Virtual machines are useful for running legacy applications that need older operating systems (for example, an application program requiring Windows 3.1 could run in a virtual machine running Windows 3.1 on a host running Windows 7 or Linux.

If the legacy system requires hardware which is significantly different from the host computer (for example, it requires a different processor or mainframe hardware), a virtual machine may not be enough, and a program called an **emulator** may be needed. An emulator is a software recreation of an entire system's hardware including its processor and associated hardware. Like a virtual machine, an emulator allows the user to run the emulated system in a window inside an existing operating system, although there may be a performance reduction.

# Legacy Systems

## Year 2000 problem

The 'Year 2000 problem' or 'millennium bug', described on page 62 highlighted problems with legacy systems which failed due to changing requirements (in this case, date calculations occurring over the end of a century).

## Comair Scheduling System

In December 2004 US regional airline Comair had to cancel all 1,160 of its flights, involving 30,000 passengers, due to record snow levels in the United States. When Comair staff tried to reschedule the flights after the weather improved, they were hit by a problem in their software, which had been programmed to only accept a certain number of changes to flights each month. Cancelling and then rescheduling over 1,100 flights exceeded that limit, and the software prevented staff from making additional changes.

The flight scheduling system involved was a legacy system created in 1986. It was written in FORTRAN and ran on a different hardware and operating system platform from all of Comair's other systems (IBM AIX rather than HP Unix). Nobody in Comair knew of the limit on the number of flight changes per month. This can be attributed partially to the fact that nobody at Comair had experience with the FORTRAN programming language, and partially because when the system was created in 1986, Comair was a much smaller company – so making such a large number of flight changes in one month was unthinkable. Comair's problem with the scheduling software is estimated to have cost it $20 million and led to the replacement of its CEO. In many ways the Comair problem was similar to the Year 2000 problem – programmers created systems which originally worked as intended, but later failed because of changing requirements.[8]

## TOPS Railway System

TOPS (Total Operations Processing System) was a system for managing railway stock and locomotives, developed by US company Southern Pacific Railroad in the early 1960s. TOPS was advanced for its time, computerising all paper records for each item of stock and allowing the records to be viewed by users with computer terminals across the country. The system was purchased and used by Canadian and British railway companies during the 1960s.

Like many systems of its time, TOPS was written in a programming language which is no longer used – in this case, a subset of the assembly language used by IBM mainframes. Finding programmers fluent in this language is not easy. Additionally, TOPS has been modified over time by its different users, and many of these changes have not been documented. The difficulty of modifying (or even understanding!) the existing TOPS code, the need for the system to be online 100% of the time, and the fact that TOPS performs its desired functions well, have contributed to it still being in use today. In some cases new user interfaces have been written to provide a more user friendly 'front end' than the original TOPS text-only display, but the underlying data is still managed by TOPS.[9]

# System development lifecycle

The **System Development Life-cycle** (SDLC) refers to the stages involved in creating an IT system – from the moment it is first suggested, to its delivery to the customer, and – critically – as it is maintained through the rest of its life (maintenance is estimated to make up 75% of a software project's cost[10]). The SDLC is critical in ensuring an appropriate system is developed, and that it meets the client's needs in terms of features, usability, cost, and time, and avoids being one of the many projects which fail (see page 316).

Various sources quote slightly different stages to the SDLC, using different names or merging certain stages together, but the key stages, which will be covered in more detail later in this chapter, are:

**Analysis:** An investigation of the current system (manual or computer), the needs of the client, and the possibility of creating a solution. After this stage it may be decided to progress with creating a new system, or that the benefits do not justify the costs.

**Design:** the planning of a solution to meet the needs of the client which were identified in the analysis.

**Development (also called Implementation):** the creation (programming) of a system, following the design previously created.

**Testing:** ensuring that the system functions correctly, as determined by the requirements generated during the analysis stage.

**Installation (also called Delivery):** installing the software and any necessary hardware, usually at the client's organisation. This may also involve removing the old system and transferring any required data from it to the new system. Training is also required to help users understand the new system.

**Maintenance:** updates and changes made to a system to fix bugs, improve performance, or add new features. In large systems which are used for many years, maintenance can form a large part of the project's total costs.

## Analysis Stage

The analysis stage of the SDLC involves investigating the current system (be it manual or computerised), determining the organisation's requirements for a new system, investigating possible solutions, determining which (if any) are feasible, and choosing the most appropriate. To be successful, the analysis must involve all key stakeholders including the **client** (the person or organisation commissioning the project) and the **end users** (the people who will use the system when it is complete) - these may be the same people. The key components of analysis are:

Determining **project goals**. The aim of project, and also its limitations (its **scope**) must be carefully defined during this phase. When a proposed project will work as part of a larger system (whether manual or computerised), it is important to know where the responsibility of each component starts and ends.

This stage is critical because a project with poorly defined goals is unlikely to be successful—the developed system may either solve a slightly different set of problems, or try to solve problems which are outside the project scope.

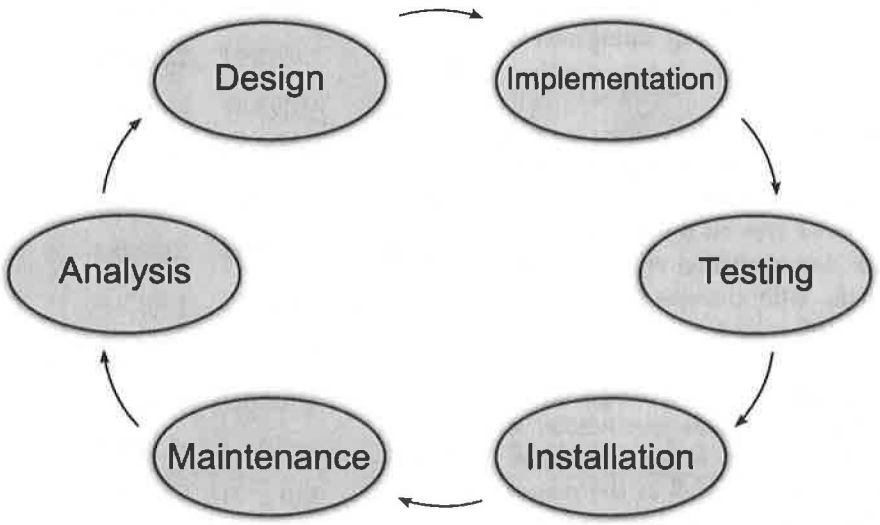



**Figure 15-2** The system development life cycle

## Project Development Roles

The **project manager** is responsible for the overall progress of a project, ensuring it stays on time, in budget, and needs the client's needs.

**Analysts**, or systems analysts, document the current system, finding its problems and areas for improvement. A **development manager** oversees **programmers** as they create the system specified in the design.

Information system managers are responsible for all IT purchases, deployments, and systems within an organisation. **Support staff** train users and help them with problems as they occur. **Database administrators** and **network managers** perform similar jobs in their specific areas.

**Data collection:** To fully understand the current system, data about it must be collected from users, managers and administrators, and any existing documentation. A variety of methods are used to gather data from users: **questionnaires** are useful when large amounts of data must be collected, but need to be carefully designed to provide accurate and useful data. Users may also provide answers they believe they are 'supposed' to give, rather than accurate answers, especially if they feel that their job may be under threat from new IT developments. **Face to face interviews** have the advantage of allowing questions to be added or changed in response to users' answers, and allow clarification to be sought on any unclear issues. However, interviews are time consuming. System developers may also **observe users** to see how they interact with the current system, the processes they use, and the output they produce. This can produce a more realistic overview of the current situation, especially if users are passively watched (for example through a video camera rather than by somebody sitting next to them).

A good source of information on the current system can be any literature related to it – **organisational policies**, **user manuals**, and **technical documentation**. However, with older systems it is possible that the system may lack documentation or that it may simply be out of date, with changes having been made but not documented.

**Requirements specification:** After investigation of the problem, a formal requirements specification is created. This is a technical document which describes the needs of an organisation as well as the project goals and scope. The requirements themselves are typically divided into two areas. **Functional requirements** cover features the system should have, including input, output, storage, and processing requirements, and the user interface. **Non-functional requirements** are limitations on how the system should work – for example, being required to run on a certain hardware platform, produce results within a certain time after user input, and be completed within a certain time period and budget—these are also known as **constraints**.

**Identification of possible IT solutions:** Many projects have several possible solutions, ranging from keeping the existing system if replacement is not feasible, to completely replacing the old system with a new IT solution. In some cases it may be viable to implement a new solution which extends the functionality of the existing solution, or works with it.

**Feasibility study and justification of solution:** At this stage, a solution has to be chosen—which may mean deciding to continue with the current system, implement one of the suggested IT solutions, or use an alternative manual system. Whatever the choice, a **business case** must be made, justifying it in terms of time and cost to design, develop, and implement, versus the predicted benefits it will bring. It must also be feasible in terms of the skills, equipment, time, and money available. Sometimes a **SWOT** (Strengths, Weaknesses, Opportunities, Threats) analysis is done to help determine the feasibility of the solution.

| Task | Start | Finish | Working Days | Responsible |
|---|---|---|---|---|
| Task A1 | 12/9/2011 | 16/9/2011 | 5 | Mr Green |
| Task A2 | 19/9/2011 | 22/9/2011 | 4 | Mr Green |
| Task A3 | 19/9/2011 | 22/9/2011 | 4 | Mr Blue |
| Task A4 | 19/9/2011 | 26/9/2011 | 6 | Mr Orange |
| Task A5 | 23/9/2011 | 27/9/2011 | 3 | Mr Green |
| Task A6 | 27/9/2011 | 30/9/2011 | 4 | Mr Blue |
| Milestone 1 | 3/10/2011 | 3/10/2011 | 1 | N/A |
| Task B1 | 4/10/2011 | 13/10/2011 | 8 | Mr Orange |
| Task B2 | 4/10/2011 | 5/10/2011 | 2 | Mr Blue |
| Task B3 | 6/20/2011 | 12/10/2011 | 5 | Mr Green |
| Finish | 13/10/2011 | 13/10/2011 | 1 | N/A |

**Figure 15-3** Table of project events and details

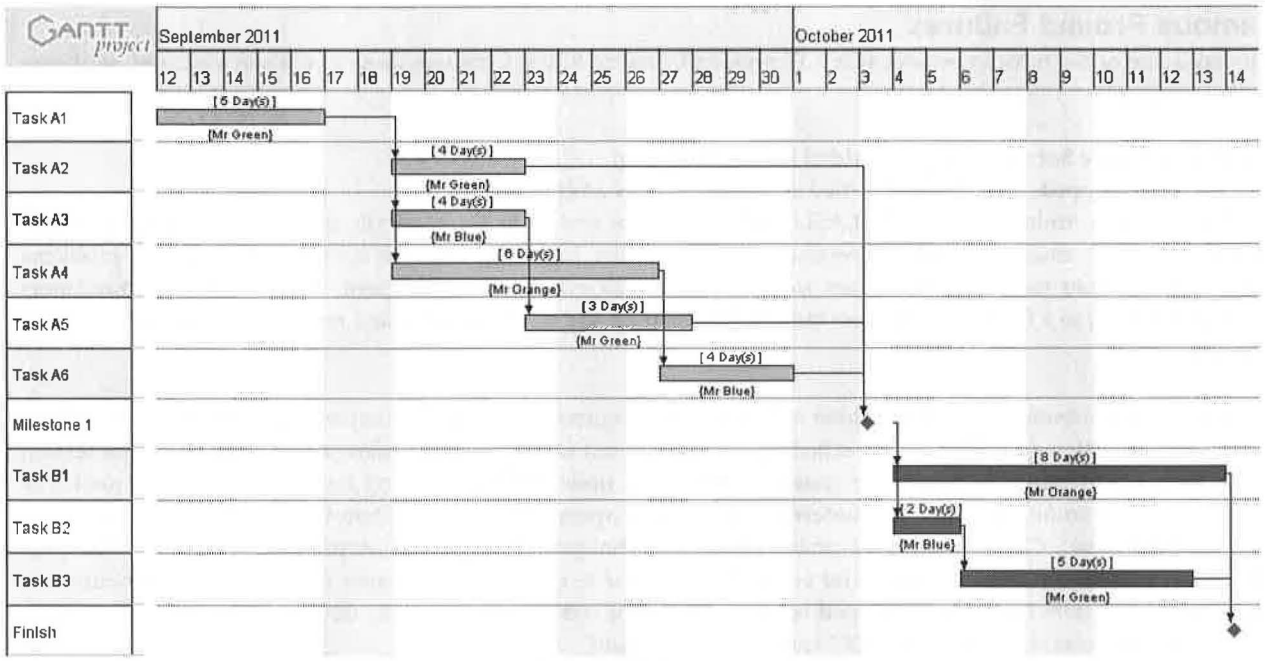



**Figure 15-4** A Gantt chart representing the tasks show in figure 15-3

**Project plan:** Assuming a project will be developed, a **project manager** is normally chosen at this point. The manager will be responsible for seeing the project through to completion, ensuring deadlines are met and the requirements specification is adhered to. A **project management methodology** will also be chosen, such as the PRINCE2, PMBoK, or one of the many methods available (see page 326). The project will be broken down into stages (which may be the SDLC stages or smaller increments of them), along with starting and finishing dates and **project milestones** – key events in the project's development process. Project scheduling information can be represented in tabular format (figure 15-3), although this can make it hard to discern any inter-task dependencies or overlapping tasks, or in diagram form.

**Gantt charts** (see figure 15-4) provide a high level overview of a project schedule, including each individual task, the people responsible for overseeing them, and task starting and finishing times. They also give a clear view of the progress which should have been made at any given time. Gantt charts can be created in specialist **project management software** or using a spreadsheet application.

**PERT (Program Evaluation and Review Technique)** charts (see figure 15-5) also represent scheduling information graphically, and clearly show the relationships and dependencies between each task in the schedule. This makes it easier to determine which activities can be worked on simultaneously, and makes it easier to spot bottlenecks in the development process. Often a process called **Critical Path** (CP) is used to determine the longest (slowest) route from the beginning of the project to the end, since this will determine the minimum time required for the project.

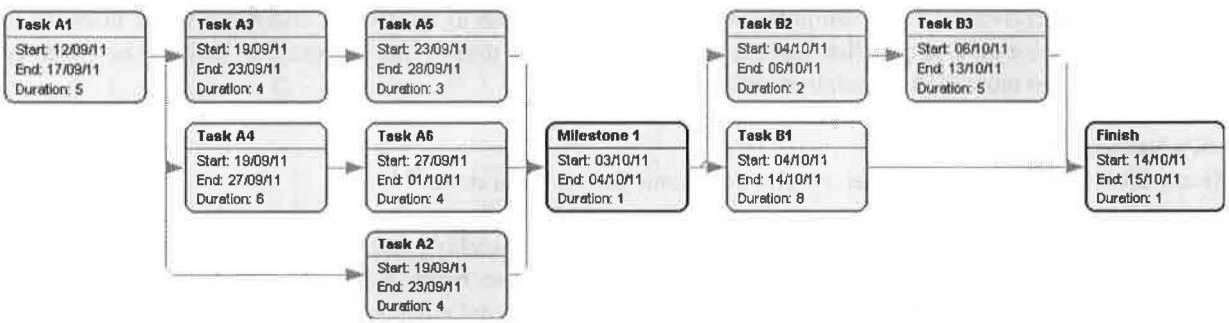



**Figure 15-5** A PERT chart showing the project schedule from figure 15-3. The Critical Path is highlighted (Tasks A1, A4, A6, Milestone 1, B1, Finish)

## Infamous Project Failures

Below are some of the most expensive failed IT projects in recent times. Compare these to the software and hardware failures on page 62.

**London Ambulance Service Computer Aided Dispatch System**

**Result:** Project scrapped, up to 30 people died as a result of slow ambulance arrival[3]

In 1992 the London Ambulance Service (LAS) introduced a new system to automatically dispatch the nearest available ambulance when an emergency call was received by its operators. Just hours after the software's installation, problems started: in some cases multiple ambulances were dispatched; in others, none were sent. In several cases ambulances were dispatched up to 8 hours late. Sources indicate between 20 and 30 people died as a result of ambulances failing to arrive in time.

Several factors contributed to the bug-ridden software: the designers were a small company with no experience of developing similar systems, and the project schedule was considered extremely short, allowing very little time for testing – particularly **load-testing**. Although the system worked well under light use during development, it was unable to deal with the large numbers of calls encountered during actual operation (and the system was never tested under these conditions before use). Compounding the problems, a direct changeover was used (see page 322), meaning staff were unable to revert to the previous system and eventually resorted to using pen and paper to keep track of ambulances. Worryingly, this system had been developed because a previous computerised system, developed in 1990, was considered unusable and was abandoned after £7.5 million of investment.

**FBI Virtual Case File (VCF)**

**Result:** Project scrapped after 5 years of development, $170 million wasted[4]

In 2000 the Federal Bureau of Investigation (FBI) started a new project to modernise its IT systems. Part of this project was the Virtual Case File software, designed to replace an existing set of programs that let FBI agents manage documents and evidence related to their cases. The system was delivered in December 2003 but the FBI considered large parts of it inappropriate or unusable. Arguments between the FBI, the developers, and Congress continued until the system was scrapped in January 2005, after more than $170 million had been spent. Even before then, the FBI was already considering purchasing off-the-shelf software to replace the failed Virtual Case File system.

A number of issues caused the failure of the project. Firstly, the initial requirements for the system were unclear, and kept changing. After the terrorist attacks on September 11th 2001, the requirements were changed from being a mere front-end to the existing systems to total migration to a new database system (the FBI's lack of information sharing capability had been a major criticism in the aftermath of the 2001 attacks). Secondly, an overly ambitious schedule was devised for both the original system and the changed system, leaving little time for testing. Finally, the personnel appointed as project managers had little or no experience of managing IT projects.

**Australian Super Seasprite Helicopter**

**Result:** $1.4 billion wasted[5]

The Australian Defence Department cancelled a contract for the new Super Seasprite helicopter in 2009 after problems with both the physical design and the software flight control system, which reportedly did not work as intended and raised fears over crew safety. The computerised system failed 4 times in 1600 hours, compared to the intended failure rate of no more than once in a million flight hours. By the time the project was cancelled, $1.4 billion had been spent - nearly $500 million more than the original budget.

**UK's National Programme for IT**

**Result:** £6.4 billion spent, project behind schedule, some features abandoned[6, 7]

Launched in 2002, the program to update the UK's National Health Service IT infrastructure has been notorious for its delays and cost overruns. The original cost estimate of £2 billion quickly grew to £12.4 billion, and the development period grew from 3 years to more than 10 years. A 2007 government report suggested opposition to the project from staff and patients was so high that the benefits of the system would not outweigh the cost. Multiple IT partners pulled out of the project, and as of 2011, parts of project have been cancelled, with others continuing on revised deadlines.

## Design Stage

During the design stage, software developers plan a solution which fulfils the functional requirements identified during the analysis stage. The design stage needs to cover, in technical detail, the **inputs**, **processes**, **data structures**, and **outputs** required by the system, and the relationship between these items.

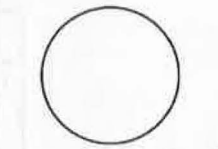
For inputs, the required items of data need to be determined, along with the data source (input from the user or an external system). The ranges of acceptable values (validation checks) must also be determined, along with the screens which accept the data from the user. Outputs, whether on screen or in the form of printed reports, are sketched to show their layout and appearance. Both input and output screens may be prototyped to test the effectiveness of their design with end users.

## Data Flow Diagrams

Often processes are designed using diagramming tools. A **Data Flow Diagram** (DFD) shows the relationship between the data storage, inputs, outputs and processes in a system, with the **data flows** between them. Internal processes and external processes (called **sinks**) are represented differently, so that the scope of the system can be clearly understood. Figure 15-7 shows the most commonly used DFD symbols.

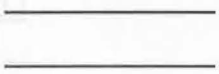
A Data Flow Diagram which shows a relatively high level view of the system, with only its relationships to external stakeholders and systems, is called a **system context diagram** (figure 15-6). System context diagrams are useful because they clearly show the boundaries of the system (i.e. what is part of the system and what is external to it) and the data that flows in and out (overall inputs and outputs). Developers usually create a system context diagram first, and then create different 'levels' of DFDs which show more a detailed view of the internal operation of the system. In complex systems, several different levels might be created for the same part of the system, each showing a more detailed view of the data flow. However, unlike flow charts, Data Flow Diagrams never show the precise details of algorithms, such as variables and decisions—only the flow of data between processes.

### DFD Symbols



**Process**. There must be at least one arrow for input and for output. The name of the process goes inside.

**Sink**. A data source or destination outside the system scope. This is 'a process performed by somebody else'. The name of the process goes inside.



**Data store**. The name of the data being stored goes inside.



**Data flow**. Arrows show the direction of flow. The name of the data should be indicated.

**Figure 15-7** Symbols used in Data Flow Diagrams (Yourdon notation)

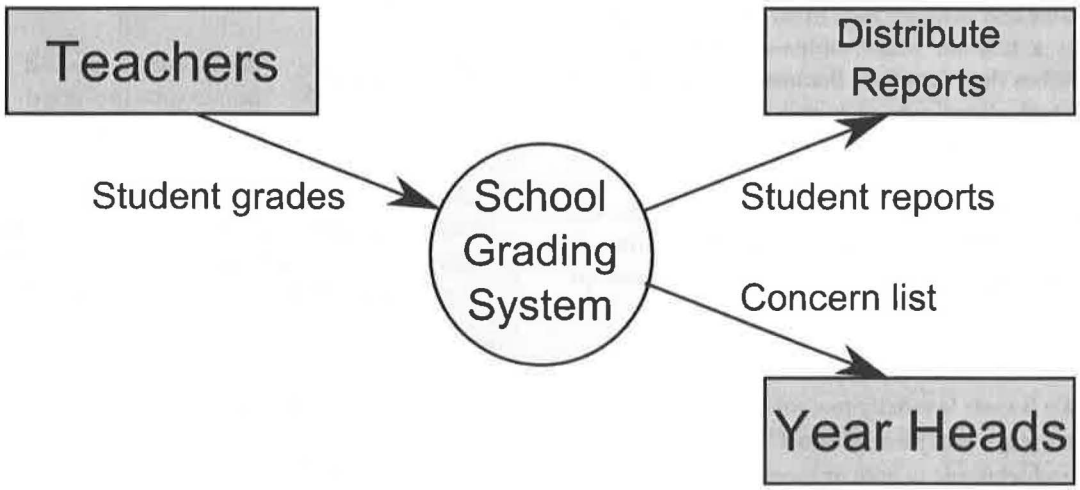



**Figure 15-6** A system context diagram which shows the relationship between the IT system (a school grading system) and external sinks. *Teachers* and *Year Heads* are clearly external stakeholders who send and receive data. Although *Distribute Reports* is a process, it is a physical task which is outside the boundary of the IT system (this would be determined by the requirements specification).

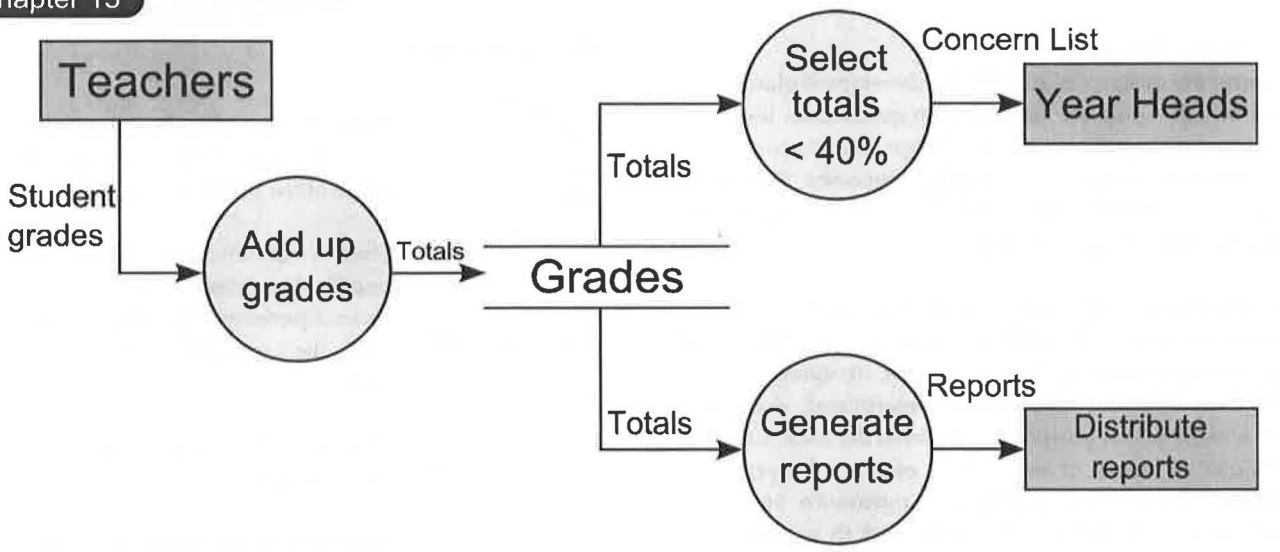



**Figure 15-8** A Data Flow Diagram showing the school grading system depicted in figure 15-7. Although there is more detail than a system context diagram, each process represented could still be further broken down.

## Entity Relationship Diagrams

**Entity Relationship Diagrams** (ERDs) are another common method of describing data storage and data relationships. An ERD shows the groups of data stored (**entities**, which represent database tables), the **attributes** each entity has (which usually map to database fields), and the **relationships** between the various data items. The **cardinality** at each end of the relationships between entities is also represented. Four options exist: zero or one, zero or more, one or more, and exactly one. Figure 15-9 shows the most common notation for ERD symbols, though several variants exist.

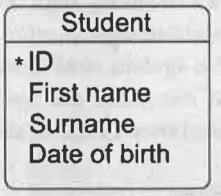
Figure 15-10 shows an ERD for a simple shop which sells products to its members. Four entities exist, each with its own attributes and primary key. In an ERD, it is convention to put a title on relationships—usually a verb—which describes their function. Because relationships can be read in both directions, each end of the relationship should be labelled in the diagram.

When reading the relationships from the diagram, the indication of cardinality closest to the start entity is skipped—so in figure 15-10 the following relationships can be read:

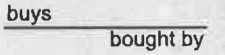
- **One** member *makes* **zero or more** sales
- **One** sale *is made to* **exactly one** member
- **One** sale *is of* **exactly one** product
- **One** product *is sold in* **zero or more** sales
- **One** product *is supplied by* **exactly one** supplier
- **One** supplier *supplies* **one or more** products

Note that the descriptions of the relationships always start with the singular 'one member....', 'one product....' followed by the name of the relationships and the cardinality of the related table.

### ERD Symbols




Entity, with the attributes belonging to that entity listed under it. The primary key is clearly indicated.




Relationship, labelled with its names (one for each direction).

**Cardinality of relationships**

Symbols on the lines indicate the cardinality (the quantity) of the relationship. Both ends of a relationships should have cardinality indicated.

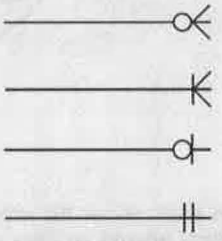



**Figure 15-9** Symbols used for Entity Relationship Diagrams

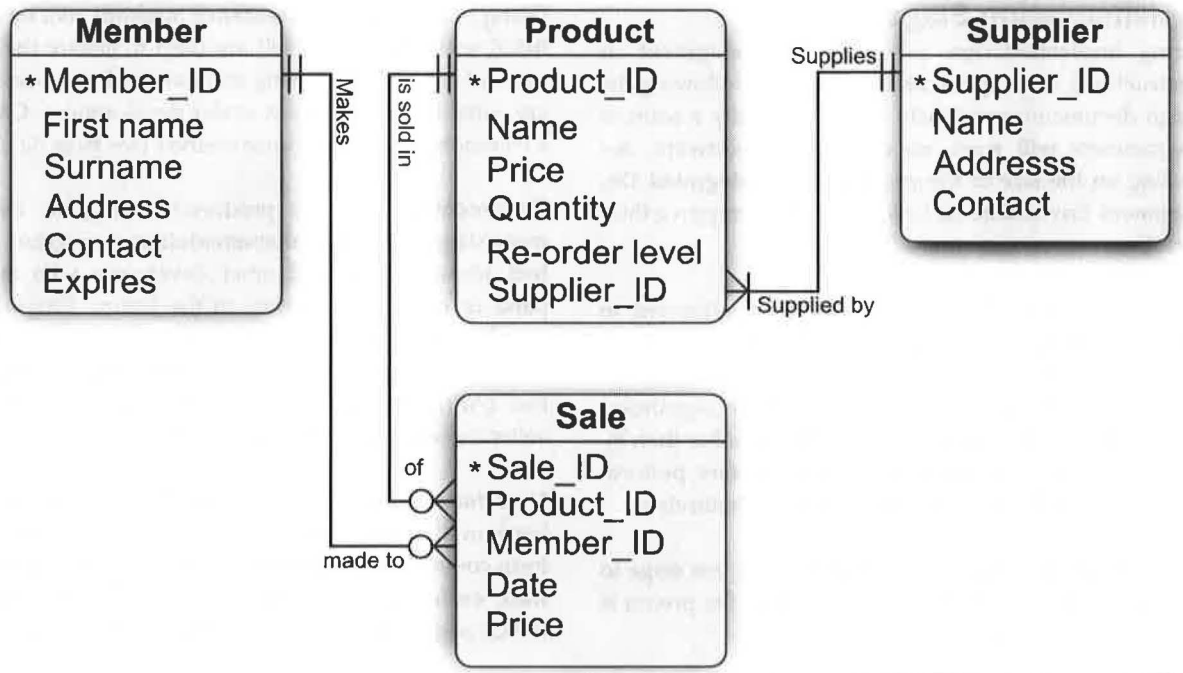



**Figure 15-10** Entity Relationship Diagram for a simple shop.

The **user interface** is also designed at this stage. This can be done using rapid development tools such as GUI builders to produce **prototype interfaces** for the user to test. Alternatively, designers may simply sketch interface designs on paper or using a computer graphics package.

The choice of user interface component, such as list boxes, drop down boxes, or text boxes, can have a significant effect on both the usability of the system and the likelihood of input errors occurring. For example, in a system with a fixed range of inputs, a drop down box takes little screen space and prevents the user from selecting invalid options, compared to a component such as a text box.

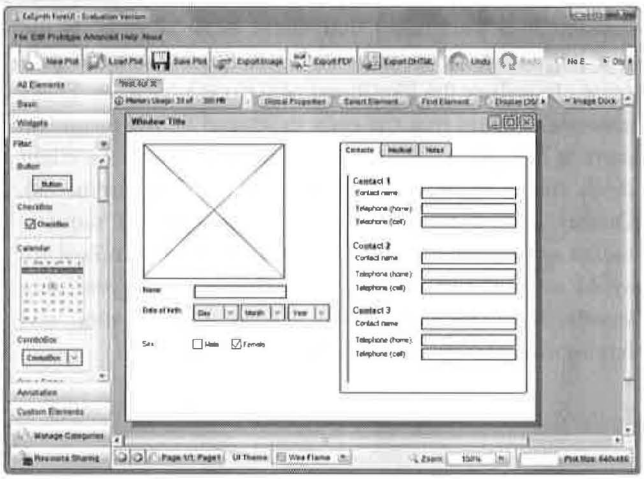


**Figure 15-11** Designers can use software tools to create prototype user interfaces

**Exercise 15-1**

Consider the two scenarios below. For each, **construct** an Entity Relationship Diagrams to show the data stored by the system, a system context diagram to show the boundaries of the system, and a Data Flow Diagram to show the movement of data within the system.

**Scenario 1:** People can rent bikes using a government scheme. They go to a bike stand and use a smart card to authenticate themselves and unlock the bike. The time, date, and location of borrowing are recorded. When the bike is returned, this information is recorded and a charge is calculated based on the number of hours borrowed. Bikes not returned after four days are considered missing and a list of their details is sent to the police. [6 marks]

**Scenario 2:** In an online shopping site, when a customer clicks 'Buy', the items held in their shopping basket are checked to ensure that they are still in stock and the prices are current. The customer is shown an order confirmation including items, price, and shipping costs. The customer enters their credit card details. These are validated with a bank. If OK, the shop sends an order to the warehouse. The warehouse packs the items and prints an invoice with the customer's address. A record of the transaction is saved. [6 marks]

## Implementation Stage

During **implementation** (also called **development** or **construction**), developers create the system, following the design documents previously created. Usually a team of programmers will work on creating the software, depending on the size of the project, using **Integrated Development Environments** (see page 310) to improve their workflow.

During implementation, **alpha testing** is performed to verify that the software works according to the requirements specification and the design documentation. At this point the software is likely to contain significant **bugs**, so the testing is performed internally rather than by customers. Usually a team of **software testers** perform alpha testing, rather than the programmers themselves.

**Prototypes** of the product may be created at this stage to demonstrate features to the client and check the project is meeting expectations.

Because it is recognised that many large IT projects fail (see page 316), developers often use **quality control processes** to reduce the likelihood of serious problems. These processes ensure that the code produced by the programmers is of high quality and meets the needs of the client (both the functional and non-functional requirements). Quality control processes include ensuring the requirements specification is written in an agreed standard (to avoid errors and omission), following programming standards to avoid common programming errors, and having comprehensive **test plans**.


**Figure 15-12** Bug tracking systems are used as part of the quality assurance process

Going further, **quality assurance methods** (not be to confused with quality control) are used to ensure the development team are following standardised practices which are suitable for the project under development. **CMMI** is a common quality assurance method (see page 321).

**Documentation** is also produced during the development stage. **Technical documentation** is targeted at system administrators and other developers who may expand or change the system in the future. Programmers usually write comments in the source code to help others who read it, and create Application Programming Interface (API) guidelines for those who will produce other software which interacts with the system.

**User documentation** explains how to use a system, covering how to install and start the system, and how to perform common tasks. **User manuals**, written or video **tutorials**, **online lessons**, and **Frequently Asked Questions** (FAQ) pages are all forms of user documentation.

## Testing Stage

Once a system has been developed and has passed alpha testing, **beta testing** can be undertaken. Versions of the software are normally given to a relatively small group of users (**beta testers**), with the aim of detecting any remaining bugs and testing the software's usability under real world conditions. Beta testing can last from a few weeks to several months. Generally beta testing only takes place once the software has been feature-frozen (no new features will be added before release).

A **bug tracking system** is used to keep a list of all software bugs (sometimes called requests for enhancement) in a central location, and allows them to be prioritised and annotated (for example, with details of how and when the bug occurs, the desired result, and the actual result). Bug tracking is an important part of the quality assurance process.

**User acceptance testing** involves the user checking the system to ensure it meets their requirements. A system may pass alpha and beta testing but still fail user acceptance testing, so it is an important part of the **handover process**. For example, part of the system may be cumbersome or slow to use – a problem not caused by a software bug, but still unacceptable to the client. Other features may be missing or not quite work as intended due to problems in the requirements specification. These issues need to be corrected before the client will formally accept the software.

## CMMI—Managing an Organisation's Maturity

**CMMI (Capability Maturity Model Integration)** is a **quality assurance method** designed to help organisations improve their performance. CMMI describes an organisation in terms of five levels of maturity (figure 15-13). CMMI applies to organisations as a whole, not just to IT projects. Models such as CMMI are important in helping an organisation build on its past experiences—building on successes and ensuring that previous mistakes are not forgotten, but learnt from, ensuring they will not be repeated.

**Level 1 – Initial Level**

At this level there is minimal organisation and planning. Management at this stage is poor, and as a result, although some things within the organisation might be done well, this is due to the initiative of individual employees and is more of an accident than the result of any formal planning. A key limitation at this level is that there is no central repository of knowledge regarding previous experience, successes, and failures. Instead, because this knowledge is only held with employees, rather than centrally, it is lost when employees move to other organisations, and the organisation is likely to repeat its past mistakes and fail to capitalise on its successes.

**Level 2 – Managed Level**

At level 2, some management takes place, allowing the organisation to track important elements such as the cost and time scale of projects, start to plan projects, and to review by comparing final performance with the plan. However, at this level these tasks still occur on a project-by-project basis – there are no central organisational standards for how these tasks should be performed. This results in variable performance across the organisation, depending on the initiative of the individuals in each project.

**Level 3 – Defined**

At this level, an Engineering Process Group (EPG) within the organisation defines standard approaches for key tasks, addressing a key problem from level 2. Employees can give feedback on these processes using a Quality Management System (QMS), so that the EPG can improve the standards over time.

**Level 4 – Quantitatively Managed**

Here statistical data and tools begin to be used in order to define goals for the organisation and thus improve its performance. For example, a goal might be to ensure all projects are delivered within two months of their deadline and within 10% of their budget. If projects fall outside of these goals, a managerial review will take place to determine whether they should continue.

**Level 5 – Optimising Level**

At the most mature level, an organisation makes small, continuous improvements to its methods – perhaps by using new tools or technologies. At level 5 weak areas are identified and improved, but these changes are more evolutionary than revolutionary.

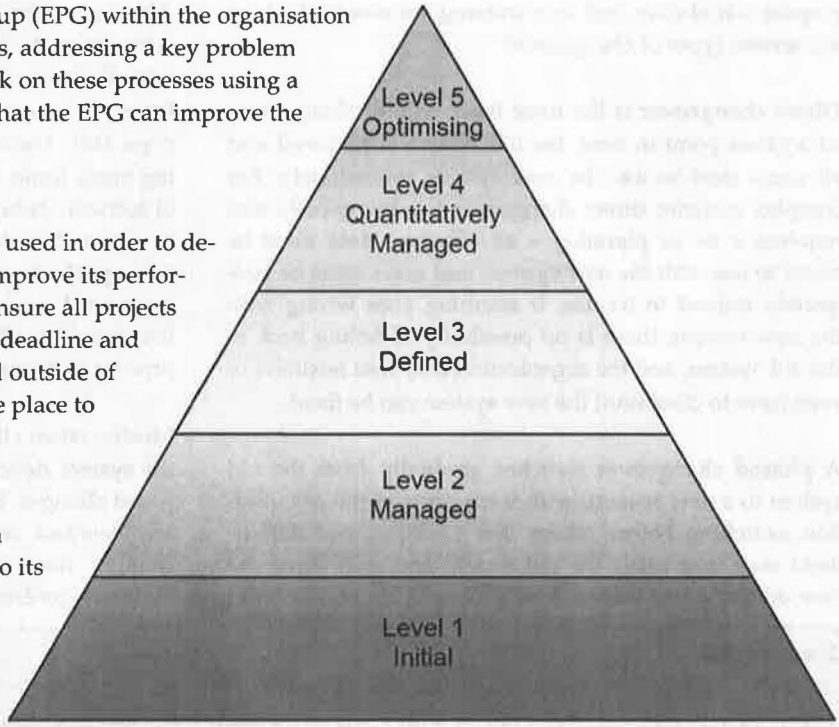



**Figure 15-13** Five levels make up the CMMI Maturity Model

## Installation Stage

Once development has finished, **installation** (also called **delivery** or **deployment**) takes place. This stage concerns preparing the organisation for the installation of the new system, the hardware and software installation, and the removal of the old system.

**User training** must be carried out before a new system can be used under real world conditions. Training may take place at the developer's location, or the new system may be installed in the client's organisation but not used for 'live' operations. User documentation plays an important part in training. Other training methods include classroom based lessons or lectures and online video tutorials. Users may be given sample data to work with in order to familiarise themselves with the software's operation.

Once training is complete, **changeover** can occur, with the old system being retired and the new system going into live use. This is a critical time for an organisation because even relatively small failures can have serious negative impacts on their business. Careful planning, preparation of data, and user training are essential. There are several types of changeover:

**Direct changeover** is the most basic type of changeover. At a given point in time, the old system is removed and all users start to use the new system immediately. For complex systems, direct changeover is a big gamble and requires a lot of planning – all required data must be ready to use with the new system, and users must be adequately trained in its use. If anything goes wrong with the new system, there is no possibility of falling back to the old system, and the organisation may lose business or even have to close until the new system can be fixed.

A **phased changeover** switches gradually from the old system to a new system, with some parts of the organisation switching before others. For example, one department may stop using the old system and start using the new one for a few weeks. If no problems are encountered after a few weeks, other departments can change over one by one. The advantage of this is that at least part of the business can still function even if the new system fails totally. However, running two systems can be complex, especially if there is a lot of interaction between departments who have changed and those who have not.

**Parallel changeover,** also called **parallel running**, involves introducing the new system but running it concurrently with the old system. This can be a time consuming method because actions must be carried out twice, and there may be significant logistical difficulties in actually using two systems side by side. For these reasons parallel running may be impractical in many circumstances. However, there are benefits for critical applications – especially safety critical systems. Running two systems simultaneously means the output of the new system can be verified against the old system and, should anything go wrong with the new system, users can immediately fall back to the old system, safe in the knowledge that it is up to date with the latest data and transactions.

## Maintenance Stage

Although often overlooked, maintenance accounts for a large amount of the time and cost associated with IT projects. Projects can take years to complete, but will be used for even longer, possibly decades (see legacy systems, page 310). Throughout its life software may need changing many times—updating it to work with new hardware or software (**adaptive maintenance**); adding new features as required by the changing requirements of the organisation (**perfective maintenance**); and fixing bugs which were not found during testing (**corrective maintenance**). **Preventative maintenance** – updating the software to prevent foreseeable problems - may also be needed.

Ideally, when changes are made to software, all stages of the system development lifecycle are applied to the proposed changes. Doing so means maintainers must update any relevant technical documentation with details of changes they make, to assist future maintainers and maintain control over the project. Software maintainers

**Exercise 15-2**

Read about the failure of the FBI Virtual Case File system (page 316). After this failure, the FBI began initial work on a new system named Sentinel in 2006, with many of the same goals as the Virtual Case File system.

Research the Sentinel project. How has its development progressed? Have the FBI learnt lessons from the failure of the Virtual Case File?

**Exercise 15-3**

Read about the London Ambulance Service project failure on page 316. **Explain** whether a direct changeover was appropriate in this instance. [4 marks]

also need to be careful to avoid causing regressions – when maintenance causes new errors in software. **Regression testing** is the process of repeating old tests to ensure they still work after alterations are made.

**Support**, although not strictly maintenance, is an important part of this process. Users may have problems using a system, which can initially be addressed with user documentation or via internal support staff. These staff provide help desk support to users using an **incident tracking system** (also called an incident **management system**). Here each support incident is recorded, along with a knowledge base of common problems and solutions. If the solution to the problem cannot be found in the knowledge base, the incident may be **escalated** and a support technician may call or visit the user. If the technician verifies the problem exists but is unable to resolve it, the incident can be further escalated, eventually being reported to the developers as a software bug. The developers themselves will use an incident tracking system to record any errors reported to them.

When a project reaches its end of life, perhaps to be superseded by a new project, it will enter the **phase out** stage, as the new project enters the analysis stage.

## Review of stages

The software development life cycle contains a large number of separate steps, which can be hard to remember. Some of these stages also have alternative names under different project management methodologies. An overview of the key stages is given below. As an ITGS student you need to be able to describe the components of each stage, explaining its role in the overall process. Where appropriate, you also need to be able to explain the different examples of each term and their advantages and disadvantages. For example, for user documentation you should be able to describe it as *documentation designed to help the user understand the new system*, give examples of user documentation such as user manuals, online tutorials, frequently asked questions lists, and explain the relative advantages and disadvantages of each of these methods.

**Analysis Stage**

- Project goals
- Scope
- Data collection
- Requirements specification
- Identification of solutions
- Business case
- Feasibility study
- Justification of solution
- SWOT
- Project plan
- Project management methodology
- Project schedule
- Project milestones

**Design**

- Inputs
- Outputs
- Data structures
- Processes
- System context diagram
- Data Flow Diagrams
- Entity Relationship Diagrams
- User interface design

**Implementation Stage**

- Development
- Alpha testing
- Prototyping
- Quality control
- Quality assurance
- User documentation
- Technical documentation

**Testing stage**

- Beta testing
- User acceptance testing
- Handover

**Installation stage**

- Installation
- Training
- Changeover

**Maintenance Stage**

- Maintenance
- Regression testing
- Support
- Incident tracking
- Phase out

## Development Approach

The way in which the SDLC stages are approached depends on the development model used. The traditional approach is to use the **waterfall model**, which flows from the analysis through each stage to the installation, one stage at a time. Analysis is carried out for the entire system, followed by design for the entire system, development of the entire system, and so on. The waterfall model is highly structured, and allows for long term planning. It ensures that problems in the analysis stage are found quickly (while they are relatively cheap to fix) rather than waiting until later stages (where they are increasing expensive to fix because each previous stage must be repeated).

However, the waterfall model lacks adaptability: large IT projects may take such a long time to develop that by the time the development stage has been reached, the requirements identified in the analysis stage may have changed (this was the case in several of the examples on page 316). This is a significant problem for the waterfall model.

The **agile development model** seeks to address some of the waterfall model's weaknesses and by allowing for greater adaptability. The same SDLC stages are encountered using agile development, but they are applied to only a small part of the problem at a time. In a relatively short period of time (usually less than a month), programmers analyse a part of the problem, design and implement a solution, and perform testing, including client **acceptance testing**. This allows working code to be produced much more rapidly, and means the client can decide whether the software is meeting their needs at every step – if it is not, it is comparatively easy to make changes, since relatively little work has been done since the previous acceptance testing.

However, a criticism of agile development is that solutions can be 'piece meal' and, for very large projects, the lack of an initial goal and design for the whole project can result in a less efficient design. In turn, this makes extending and maintaining the project later on much more difficult.

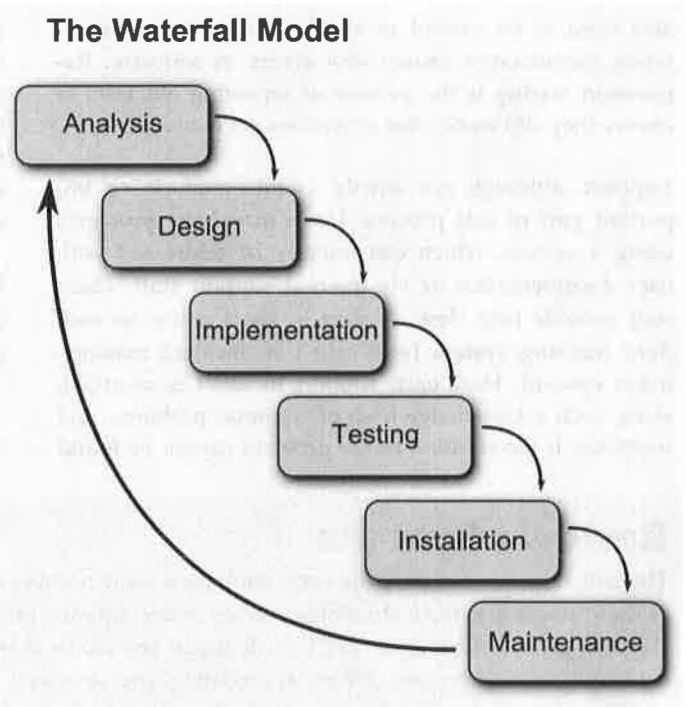



**Figure 15-14** The traditional Waterfall model of software development, in which each stage is completed before moving onto the next.

**Exercise 15-4**

A software development company is planning to create a School Management Information System (MIS), including features for storing student personal details (names, contact details), reporting of grades to parents, teacher entry of grades, timetabling / scheduling of lessons, and storing medical details for the school nurse. The project is scheduled to begin development in July 2011 and be complete ready for use in June 2012.

Use appropriate software to:

a) **Construct** a Gantt chart showing the project schedule if a waterfall model was used. [6 marks]
b) **Construct** a Gantt chart show the project schedule if the agile model was used. [6 marks]
c) **Construct** a PERT chart based on the Gantt chart created in part (a) [6 marks]

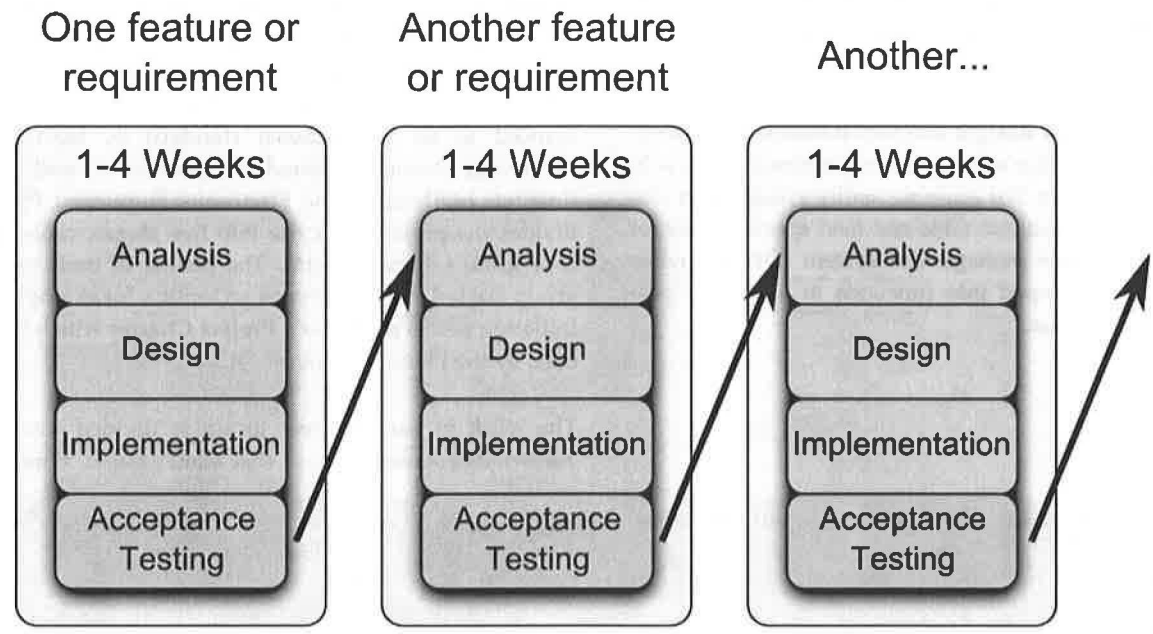



**Figure 15-15** The agile model of software development breaks development down into shorter time periods

## Project Management Methodologies

Good project management is essential for success. The SDLC describes the steps in project development, and the waterfall model and agile model describe the order in which development is approached. **Project management methodologies** aim to describe the best approaches for managing those steps, moving between them, or recording successes and failures to enable future improvement.

### SSADM

**SSADM (Structured Systems Analysis and Design Method)** is a methodology which focuses primarily on the Analysis and Design stages of the SDLC. As a result, many of the items and activities in SSADM are similar to the activities described on pages 313 to 322, though they have slightly different names. SSADM describes 7 stages:

**Stage 0 - Feasibility study:** examines whether an IT project is feasible, in terms of costs, equipment (hardware and software), and existing organisational systems. This stage examines various possible solutions, and if they have been rejected as infeasible, records why.

**Stage 1 - Investigation of the current environment:** includes creating a context diagram, DFD and ERD diagrams for the current system, plus a list of users and how they interact with the system. Problems and inadequacies of the current system will be documented in detail. It is essential to thoroughly understand the current system, so limitations and opportunities can be identified. This is important because a new system should be based on business needs – not merely a re-implementation of the current system (which may have limitations).

**Stage 2 - Business system options:** At this point the analyst presents different options for solving the business' problems. This could include continuing to use the existing system. A cost / benefit analysis of proposed systems needs to be completed too, as well as examining any negative impacts the new system may have.

**Stage 3 – Requirements Specification:** a detailed list of requirements is generated. Using the understanding of the current system from the investigation in stage 1 as a basis, DFD and ERD diagrams are constructed to represent the new system. User roles are also defined.

**Stage 4 – Technical system options:** involves considering the hardware, software, networks, and personnel that might be required for the proposed system. As in stage 2, it is likely that several options will be presented, with the rejected options being recorded.

**Stage 5 – Logical design:** Involves identifying, modelling, and documenting all of the required data for the system. This involves diagramming work using tools such as enti-

ty relationship diagrams and data flow diagrams to model the proposed system. This stage is largely analogous to the design stage in the waterfall model of system development.

**Stage 6 – Physical design:** involves translating all previous design work into actual physical hardware and software specifications. For example, entity relationship diagrams are translated into table and field specifications for a specific database management system (DBMS), while processes are mapped into functions in a specific programming language.

## PMBoK

**PMBoK (Project Management Body of Knowledge)** is not specific to software development projects – its techniques can be used to manage virtually any project, from building a house to designing an aircraft. PMBok is recognised as an international standard by both ANSI (American National Standards Institute) and IEEE (Institute of Electrical and Electronics Engineers). PMBoK divides the project life cycle into five stages, called **process groups** (figure 15-16). The results of each process group are fed into the next as an input – for example, the Initiating group produces a **Project Charter** which is then used by the Planning group.

The work in each process group is divided into nine **Knowledge Areas**, such as cost management, time man-

| Process Groups / Knowledge Areas | Initiating | Planning | Executing | Controlling and Monitoring | Closing |
|---|---|---|---|---|---|
| Integration Management | Develop Project Charter | Develop Project Management Plan | Direct and Manage Project | Monitor and control work | Close phase or project |
| Scope Management | | Collect requirements<br>Define scope<br>Create WBS | | Verify and control scope | |
| Time Management | | | | | |
| Cost Management | | Estimate costs<br>Plan budget | | Control costs | |
| Quality Management | | | | | |
| Human Resource Management | | | | | |
| Communications Management | Identify stakeholders | Plan communications | Distribute information and manage expectations | Performance report | |
| Risk Management | | | | | |
| Procurement Management | | Plan procurements | Conduct procurements | Administer procurements | Close procurements |

**Figure 15-16** An overview of PMBoK (only certain key cells have been filled in)[2]

agement, and communications management. Each of these represents an area of specialisation within the project, and has specific tasks during each process group. For example, during the **Initiating** stage, the *Communications Management* should identify the stakeholders. During the **Planning** stage, the *Communications Management* should plan their communication with the stakeholders, and during the **Executing** stage, they should actually communicate with the stakeholders. Key knowledge areas include **Scope** management, which determines the features and limitations of the project, and **procurement** management, which deals with acquiring any new hardware, software, and personnel required for the project.

**Key Point**

As an ITGS student you do **not** need to remember the names of each process group, knowledge area, all of the 42 PMBoK tasks, or be able to draw a diagram of the PRINCE2 processes.

It is sufficient to understand the purpose of these project management methodologies, the approaches they use, and some of the key events and features that each methodology has (for example, the Project Initiation Documentation in PRINCE2).

## PRINCE2

**PRINCE2 (PRojects IN Controlled Environments 2)** is the standard project management methodology used by the UK government. PRINCE2 starts by considering a **business case** for a project – i.e. an evaluation of the benefits and the challenges. Planning, organisation, and risk management are key features of the PRINCE 2 approach.

Like PMBoK, PRINCE2 divides the life cycle into stages – in this case, **Starting Up**, **Initiating**, **Delivering**, and **Final Delivery**. Each stage is broken into levels – **Directing**, **Managing**, and **Delivering** – which describe the type of work performed (figure 15-17). At the Directing level, the entire project is directed, typically by corporate management. At the Managing level, **project managers** guide parts of the project, and transition between stages (this is known as **Managing Stage Boundaries**). At the Delivery level, the project is created (in IT development, this would be the programming).

Notable features of PRINCE2 include the Project Initiation Document, which contains the results of the analysis tasks (project goals, scope, constraints, and so on).

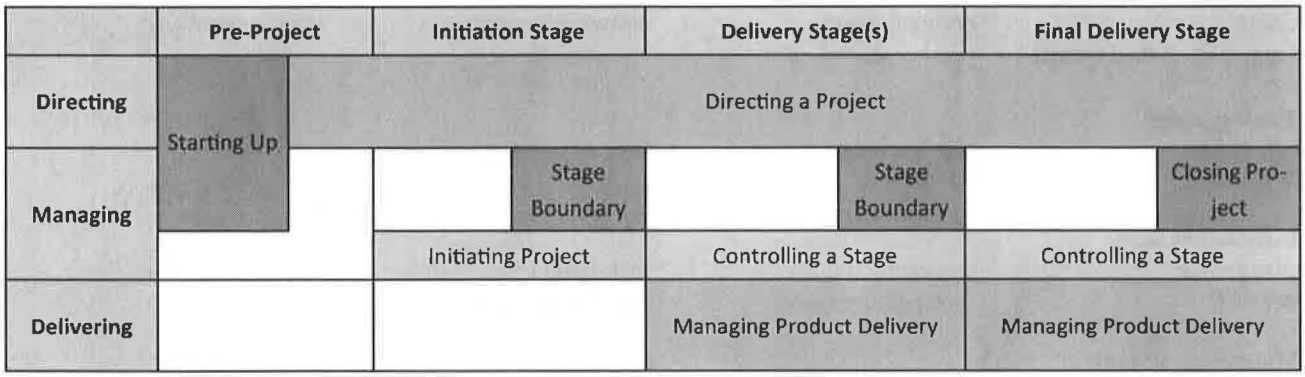



**Figure 15-17** Overview of PRINCE2

## Professional Conduct and Ethics

The **Association for Computing Machinery** (ACM) and the **British Computer Society** (BCS) both produce **codes of conduct** for IT professionals. These cover best professional practices and provide guidelines for ethical conduct. Stipulations in the ACM's code include working to 'Contribute to society and human well-being ', 'Respect the privacy of others', and 'Honor confidentiality'[11] - all important traits when working on large scale IT projects. Other responsibilities in the code include respecting intellectual property and copyright laws.

Particularly relevant to the development of large IT projects, items such as 'Acquire and maintain professional competence', 'Know and respect existing laws pertaining to professional work', and 'Accept and provide appropriate professional review'[11] put the onus on developers to create appropriate systems for their clients, using the their skills to the best of their ability and—critically—not to undertake projects which are beyond their qualifications, capabilities, or experience. This last point in particular has been the cause of several IT project failures—most notably, the London Ambulance Service dispatch system (see page 316), which was created by developers with no experience in the field, and Therac-25 (see page 63), whose code was created by a lone programmer.

# Chapter Review

## Key Language

**IT Roles**

information system managers
analyst
database administrator
development manager
network manager
programmer
support staff

**Analysis Stage**

business case
client
constraints
Critical Path
data collection
end-user
feasibility study
functional requirements
Gantt chart
milestones
non-functional requirements
organisational IT policies
PERT chart
project goals
project management methodology
project management software
project manager
project plan
requirements specification
scope
SWOT
technical documentation
user documentation

**Design Stage**

attributes
cardinality
data structure
DFD
entities
ERD
outputs
processes
prototype interface
relationships
system context diagram
user interface

**Implementation Stage**

alpha testing
bug
CMMI
Frequently Asked Questions
implementation
load testing
online lessons
prototypes
quality assurance methods
quality control processes
software testers
test plan
training
tutorial
user manual

**Testing Stage**

beta tester
beta testing
bug tracking system
handover
user acceptance testing

**Installation Stage**

changeover
delivery
deployment
direct changeover
installation
parallel running

**Maintenance Stage**

adaptive maintenance
corrective maintenance
incident escalation
incident management system
tem
incident tracking system
internal support
maintenance
perfective maintenance
phase out
phased changeover
preventative maintenance
regression testing
support
training

**Development Approach and Methodologies**

acceptance testing
agile development
PMBoK
PRINCE2
procurement
project initiation document
SSADM
user acceptance testing
waterfall development

**General Terms**

code of conduct
custom / bespoke software
emulator
IDE
legacy system
off-the-shelf software
source code
system development lifecycle
virtual machine

**Exercise 15-5**

**Construct** a system context diagram and an Entity Relationship Diagram for the following scenario:

Patients make appointments with doctors at a hospital. Patients are billed separately for each appointment they attend. Some patients pay their bills at the time of the appointment; others have insurance companies. Hospitals keep a record of patients' personal details, the appointments they have attended, and any treatments given at each appointment. Some doctors have one or more specialities, and patients may be referred to these doctors for treatment.

**Exercise 15-6**

**Construct** a Gantt chart using the project schedule data in the table below.

| Stage | Start Date | End Date | Responsible |
|---|---|---|---|
| Analysis | 1 Sept 2011 | 14 Nov 2011 | Jane Pear |
| Design | 15 Nov 2011 | 16 Feb 2012 | Lucy Riess |
| Development | 22 Nov 2011 | 25 Mar 2012 | Sam Tallen |
| Testing | 5 Jan 2012 | 25 Apr 2012 | Jack Patiol |
| Acceptance | 1 May 2012 | 14 May 2012 | Clarence West |

**Exercise 15-7**

(a) (i) Define the term *beta test.* [2 marks]

(ii) Describe **two** steps that occur as part of the Analysis stage of the SDLC [4 marks]

(b) Explain how CMMI helps ensure project quality. [6 marks]

(c) To what extent is the choice of the Agile model of system development better than the choice of the Waterfall model? [8 marks]

**Exercise 15-8**

A large haulage company has a legacy system for managing its inventory. The system was developed 15 years ago and runs on aging hardware with a text based user interface. The company wants to expand their system to allow new features, including giving customers access to their data over the Internet. They are considering the purchase of a new system to achieve these goals.

(a) (i) Define the term *technical documentation.* [2 marks]

(ii) Describe **two** types of changeover that could be used with this system. [4 marks]

(b) Explain **two** significant challenges faced in developing a custom / bespoke application [6 marks]

(c) To what extent is it appropriate to develop an entirely new system to replace the legacy system? [8 marks]

**Exercise 15-9**

A restaurant wishes to implement a computerised system to replace its current paper files for maintaining records of sales, purchases, suppliers, and staff. It is considering whether to use a commercial off the shelf application or commission a custom / bespoke application.

(a) (i) Define the term *legacy system* [2 marks]

(ii) Distinguish between a project's *client* and the *end user.* [4 marks]

(b) Explain **two** methods that can be used to train users on a new system. [6 marks]

(c) Evaluate the choice of a commercial off the shelf application rather than a custom / bespoke application for a business. [8 marks]

## References


1 McKendrick, J. (2011). *Study: US government spends $36 billion a year maintaining legacy systems.* ZDNet. Available: www.zdnet.com/blog/service-oriented/study-us-government-spends-36-billion-a-year-maintaining-legacy-systems/6505?tag=nl.e019. Last accessed March 2011.

2 Project Management Institute (2004). *A Guide to the Project Management Body of Knowledge.* Project Management Institute.

3 Lancaster University. (Date unknown). *Case study : The London Ambulance Service Despatching System.* Available: http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/CaseStudies/LondonAmbulance/index.html. Last accessed Nov 2011.

4 Goldstein, H. (2005). *"Who killed the Virtual Case File?".* IEEE Spectrum. Available: http://spectrum.ieee.org/computing/software/who-killed-the-virtual-case-file. Last accessed March 2011.

5 Walters, P. (2009). *$1.4bn wasted on cancelled Seasprite.* Available: www.strategypage.com/militaryforums/512-48220.aspx. Last accessed March 2011.

6 Hope, C. (2007). *Patients 'won't benefit from £12bn IT project'.* The Telegraph. Available: www.telegraph.co.uk/news/uknews/1548813/Patients-wont-benefit-from-12bn-IT-project.html. Last accessed March 2011.

7 National Audit Office. (2006). *The National Programme for IT in the NHS.* Available: www.nao.org.uk/idoc.ashx?docId=01f31d7c-0681-4477-84e2-dc8034e31c6a&version=-1. Last accessed March 2011.

8 Overby, S. (2005). *"Comair's Christmas Disaster: Bound To Fail". CIO.* Available: www.cio.com/article/112103/Comair_s_Christmas_Disaster_Bound_To_Fail. Last accessed March 2011.

9 TrainWeb. (Unknown date). *The History of TOPS.* Available: www.trainweb.org/rews/tops/history.htm. Last accessed March 2011.

10 Clarity in Code. (2011). *Software Maintenance.* Available: http://www.clarityincode.com/software-maintenance/. Last accessed Nov 2011.

11 ACM. (1992). *ACM Code of Ethics and Professional Conduct.* Available: http://www.acm.org/about/code-of-ethics. Last accessed Nov 2011.