development, since the cost of development has been spread over a number of users. The software can be examined and tested prior to purchase, minimising the risk involved, and user training and support are often commercially available. Even if some customisation is required, a package can be installed in a very much shorter time frame than if an equivalent system is developed from scratch. In general, if a suitable package is available, then this is the best option for acquisition of software.

11.3.3 Outsourcing

Outsourcing involves the purchasing of a service, in this case software development, from another company. With rapid changes in technology and high turnover rates for IS staff, it becomes increasingly difficult to maintain high levels of expertise in-house. Outsourcing allows an organisation to focus on its core competencies, while benefiting from the expertise and economies of scale that can be provided by the outsourcing company. Contracts and service level agreements need to be clearly defined, and in many cases organisations will outsource only their more general systems, while retaining control over strategic projects. In this case, additional contract staff can be used to provide skills that are not available internally.

A recent trend is towards the use of ***application service providers*** (ASPs), who maintain a range of software programs on their own computers. These are made available to customers via a web interface, and are a cost-effective solution for the provision of software that is used infrequently or from varying locations. The principle underlying this approach is that just as a company would not consider producing electricity privately to meet its own needs, but finds it easier to pay a supplier based on usage, so the company can pay for the use of applications which have been developed, installed and maintained by an ASP.

11.3.4 End-user computing

With the growth of computer literacy in the workplace, plus decreased costs of computer hardware and user-friendly development tools, ***end-user computing*** has become an important factor in system development. Faced with frustrating backlogs in the IS department, or inspired by the opportunities presented by new technologies, an increasing number of users are attempting to develop their own systems. The majority of user-developed applications tend to be personal or departmental in function, and often do not adhere to organisational standards. Data validation and security may be poor, backup and recovery procedures are often missing, and lack of documentation means that if the developer leaves the organisation, then the system is likely to fall into disuse. In response to these concerns, user support centres may be established to provide training and assistance for users.

11.4 Project Management

Project management differs from general business management in that a project has a beginning and an end, between which a definable set of activities must occur. Most projects are once-off efforts, so previous experience is likely to be limited, and the high degree of interaction between participants and tasks adds to project complexity. In order to meet these challenges, special project management tools have been developed to assist with project planning, resource allocation, scheduling and review.

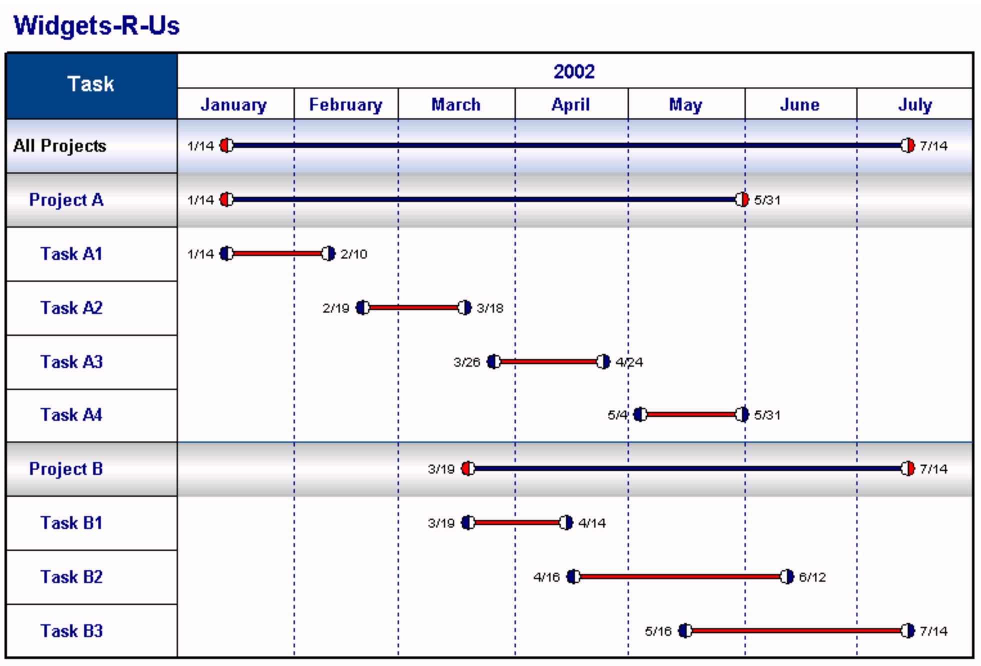

Figure 11-3. Example of a Gantt chart

The first task of the project manager is to define the scope of the project, the tasks to be accomplished and the order in which they should be done. This information can be represented using a ***Gantt chart*** (see Figure 11-3), which also reflects the start and end dates of major tasks and the staff involved. Dependencies between tasks then need to be identified, since any hold-up at a critical point might affect the completion time of the project as a whole. A ***network diagram*** or ***Pert chart*** shows the minimum and maximum time that might be needed for each task, as well as which tasks must be completed in sequence and which can be addressed in parallel, so that the impact of delays can be accurately assessed. ***Risk assessment*** techniques can be used to identify risk factors and implement strategies for minimising the potential problems associated with systems involving complex technology or where user requirements are not clearly defined.

The progress of a project must be monitored throughout its lifespan, usually on the basis of *deliverables* submitted at the end of each phase. Comparisons with the budgeted time and cost are made at frequent intervals, so that problems can be identified and corrected (or the project abandoned) as early as possible.

Another important element in project management is the acquisition and training of staff and the development of relationships both within the team and with the future users of the system. This can be supported through the organisation's intranet together with the effective use of groupware applications such as email and electronic meeting support.

It should be apparent from these examples that more than one software tool is required to provide the different perspectives needed for effective project management. Many of the same tools can also be used to improve general business efficiency, for example in the periodic scheduling of jobs and employees by operations managers.

11.5 People Aspects of Systems Development

The systems development process will involve IS professionals, management, end users and outside vendors.

Management. Decisions as to which computer systems are to be developed and the allocation of IS resources to current projects must be made by general management with advice from the IS department. One way this can be achieved is to form a computer steering committee with representatives from top management, managers of the various business functions and the IS manager. The *chief information officer* will manage IS resources (including staff), liaise with top management and advise on strategic opportunities.

Historically IS developers have a poor record in terms of delivery of new computer systems on time and within budget. The appointment of a *project manager* (who may be a systems analyst with the required project management skills) will help to keep the project on schedule.

IS professionals. The *systems analyst* plays the part of an intermediary in the development process, providing the link between users and the programmers and technicians who will build the system. The analyst’s role is to develop the user requirement and detailed design specifications and to provide continuity and management support throughout the project. He or she must therefore have a strong understanding of the business together with the required technical skills. In addition the analyst must possess the management and behavioural skills to supervise and communicate with people.

In many organisations the analyst’s role has been split into that of the *business analyst*, normally an individual with a strong commercial background who will perform the analysis function, and the *systems designer*, a technical analyst who will develop the detailed specifications of the system.

Programmers work from the detailed program specifications developed by the analyst, and code and test the required program modules.

In large organisations there are individuals in technical roles who will provide support for development projects in the design and implementation stages. For example the database administration function will provide expertise in designing and construction of the database, network specialists will assist in the design and implementation of local or wide area communications, and the web administrator will be responsible for the development, delivery and integration of web resources.

Internal auditors will be needed to review the controls of the system.

End-users. These are the people who will ultimately use the new system. In the past they were largely ignored once the analyst felt their requirements were known. Modern practice dictates that users be involved in every stage of the development project. This involvement has two main objectives. Firstly the user can define what he wants and give details of the user interface. In addition user involvement breaks down resistance to change. Making the user part of the development team ensures a commitment to the successful implementation of the application.

In many cases, end-users are also assuming a role as system developers, although the responsibility for network, database and hardware management is likely to remain under the control of a centralised IS department.

Outside vendors. Outside organisations may be providing hardware, software or their consulting expertise. Many organisations use outside contractors as analysts or programmers to help in development projects. Contractors are useful when an organisation lacks technical skills in an area or has a shortage of staff. The problem with contract staff is that the knowledge and skills developed during the project can disappear when the contractor leaves. Adequate documentation and training of in-house personnel are the obvious solutions.

11.6 South African Perspective

Clover Dairies, the oldest and largest dairy company in South Africa with over 6500 employees and a turnover exceeding R3 billion, distributes its products across a widespread customer base. Early in 2003 they invested in the Heat customer service system to support centralised information requests, order processing, claim handling, invoice and credit note queries, complaints, account administration and customer requests, as well as conducting customer and product surveys. According to the ICT systems manager at Clover, the decision to buy this particular product “was based on several criteria, including cost-effectiveness, product-effectiveness, the strategic importance of the product to Clover, its functional fit, and the levels of service and support available”. Heat will also be integrated into Clover’s existing business planning and control systems as well as its telephony system, and will be accessible via a web interface over its network connection.

11.7 Beyond the Basics

There are various issues to be considered when planning for e-commerce applications.

- Rather than the traditional cost-benefit approach, web initiatives may be seen as a strategic investment to be used as a platform for future business activities.
- As part of the overall information architecture of the organisation, a top-level decision must be made as to whether the entire site will be developed and maintained as a single project (either in-house or outsourced), or whether only core functions will be initially planned and developed, allowing departments to extend it later as required.
- An extended group of users needs to be considered in planning a web-based system: surfers who are browsing the Web, customers who wish to make online purchases, suppliers and recipients. Many of these are not usually regarded as direct users of traditional IS applications. Accessibility issues are also raised, to allow for the possibility of external users who are physically or mentally handicapped.
- Even though the benefits resulting from a website are difficult to quantify, the effectiveness of the site in fulfilling its purpose must be evaluated in the same way as any other IT investment, to ensure that business requirements are being met and problems are identified.

11.8 Exercises

11.8.1 Value chain

For each stage of the (primary) value chain, how could IS be used to cut costs or add value to business activities?

11.8.2 Software acquisition options

The different software acquisition options available to organisations include in-house development, commercial packages, outsourcing and end-user development. Compare these three approaches in terms of each of the following factors:

- **Cost control**: how well can costs be estimated at the start of a project, and how successfully can budgets be enforced?
- **Availability of expertise**: how likely are you to run into difficulties if the project turns out to be more complex than was initially envisaged?
- **Quality of the final system**: can you be sure that the final system will meet your requirements, and will do the job without errors?
- **Documentation and training**: will these be available when the system is first used, and at later stages if required (e.g. for new staff)
- **Maintenanc**e: how easy will it be to make changes to the system in five years time? In ten years time? What if you need to integrate other systems with this one?

12. System Development

This chapter will provide you with an overview of the systems development process. First we describe in detail the traditional **Systems Development Life Cycle** (SDLC), encompassing the stages through which each system should pass, from the initial survey to hand-over of the completed system.

Pressure for rapid development and future maintainability of systems has resulted in a number of alternative approaches to systems development, ranging from development by end-users, to the incorporation of formal methods to improve the quality and efficiency of the development process.

12.1 Systems Development Life Cycle (SDLC)

Systems development could be seen as the simple process of writing programs to solve the needs of the user. Unfortunately the user knows what he wants but has no technical expertise while the programmer understands the computer but not the user environment. This communication gap between the customer and the service provider must be handled by an intermediary, the systems analyst. Broadly speaking therefore the systems analyst translates user's needs into detailed specifications for implementation by the programmer.

Over the years the software manufacturing process has become more formalised:

> "The basic idea of the **systems development life cycle** is that there is a well defined process by which an application is conceived, developed and implemented. The life cycle gives structure to a creative process. In order to manage and control the development effort, it is necessary to know what should have been done, what has been done, and what has yet to be accomplished. The phases in the systems development life cycle provide a basis for management and control because they define segments of the workflow which can be identified for managerial purposes and specify the documents or other deliverables to be produced in each phase." [Davis and Olson, 1985]

The number of stages and names to describe those stages differ slightly between organisations; but the SDLC normally covers the activities shown in figure 12-1, each with a primary purpose.

12.1.1 Preliminary Investigation

The **preliminary investigation** is carried out to determine the scope and objectives of the new system and to investigate whether there is a feasible solution. New applications normally originate from end-user requests and are weighed against the other requests for IS resources before approval to develop the system is granted. At this stage an analyst or small project team is authorised to investigate the real potential of the new application. During this brief study the analyst must investigate the problem and the existing system sufficiently to be able to identify the true extent and purpose of the new application.

In order to ensure that the new system would be of greater benefit to the organisation than other competing requests for proposals, a **feasibility study** must be performed covering the following three major areas:

- Economic feasibility to measure the costs and benefits of the new system.
- Technical feasibility to ensure that the organisation has sufficient hardware, software and personnel resources to develop and support the proposed system.
- Operational feasibility, the willingness and ability of management, users and Information Systems staff in the organisation to build and use the proposed system.

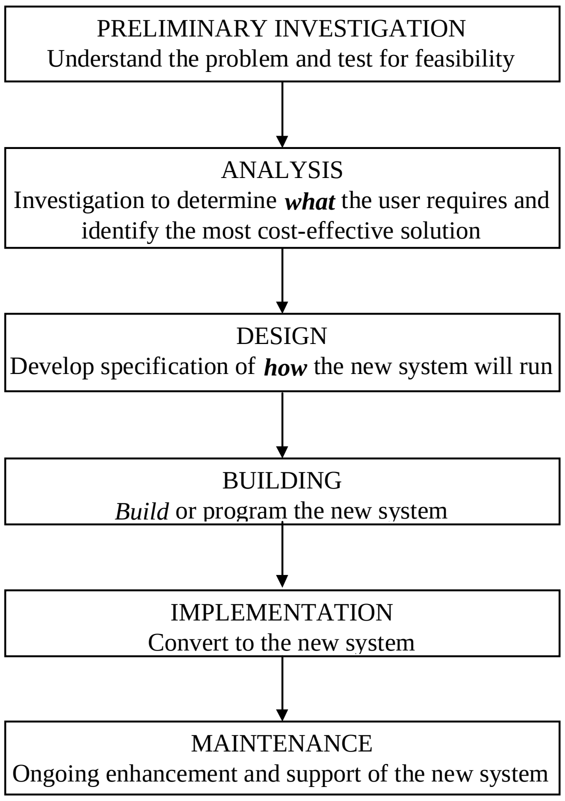
PRELIMINARY INVESTIGATION
Understand the problem and test for feasibility
ANALYSIS
Investigation to determine *what* the user requires and identify the most cost-effective solution
DESIGN
Develop specification of *how* the new system will run
BUILDING
Build or program the new system
IMPLEMENTATION
Convert to the new system
MAINTENANCE
Ongoing enhancement and support of the new system

Figure 12-1: Systems Development Process

Issues such as the size and complexity of the new system and the skills and availability of user and IS staff, will determine the level of potential risk to the organisation in developing the system.

The output from this preliminary investigation is a statement of scope and objectives (often termed the **project charter**) together with a feasibility report. This document is submitted to management where a decision is made as to whether or not the development project should continue.

12.1.2 Systems Analysis

In this stage the analyst investigates the needs of the user and develops a conceptual solution to the problem. One human failing we all tend to exhibit is to rush into proposing solutions before we fully understand the problem we are trying to solve. It is therefore important for the analyst to develop a broad, conceptual solution to the problem (what needs to be done) prior to launching into the detailed physical design where we specify how the system will work.

In the past analysis tended to be very much a pragmatic affair with success more dependent on the experience and capabilities of the analyst than on any formalised approach. The analysis phase should include the following discrete steps:

- **Understand** how the existing system operates. This information can be obtained by observing people at work, interviewing users, and studying procedure manuals and other supporting documentation, questionnaires and visits to other organisations.

- **Document** the current physical system. A major problem in the past was how to record all the detail about the system. Most of it could be found only in the analyst's head or draft notes. Here the basic tools of structured systems analysis such as the data flow diagram (DFD), the entity relationship diagram (ERD) and data dictionary (DD) can be used to represent graphically and record the data and procedures. We will discuss these later in the chapter.

- **Define** the problem areas. These may include such issues as poor response times in the existing system, poor presentation of current information, high costs or weak controls in the current system, waste and sometimes duplication.

- **Identify** new requirements. The analyst must attempt to identify new user requirements and look for new and improved procedures that can be incorporated into the system.

- **Identify** possible solutions. Having derived objectives for the new system from the previous stage, the analyst now develops a conceptual model of the new system in conjunction with the user. This may involve the investigation of alternative physical designs, such as whether to remain with the existing manual system, or to choose a centralised or decentralised approach to the application.

- The culmination of the analysis stage is the preparation of the formal **user requirement specification** (URS) that incorporates a logical model of a system that will meet the user's requirements. A large proportion of the functional description and data specifications is best communicated from the analysis stage to the design stage through the graphic and electronic output from the structured tools used in the analysis process (data flow diagrams, entity relationship models, decision trees and the data dictionary).

- Again management is required to **review** the status of the project and to make its go/no-go decision.

12.1.3 Systems Design

The analysis stage of the SDLC has clearly identified what must be done in order to meet the user's requirements. One important decision that must be taken at this point is whether to "*make or buy*" the new software application. In the past, most large organisations developed their own applications as no two organisations were exactly alike, and they could afford the investment in systems developed around their user's needs.

Today the picture is changing as custom-built software is becoming very expensive to develop and even more so to maintain. Computer applications are large, complex and integrated and many businesses have become non-competitive because of their inability to develop systems that adequately support their business activities.

On the reverse side, **packages** (pre-written software applications) are becoming more common and can be customised to meet the needs of each organisation. With major benefits in terms of speed of installation, cost, low maintenance and low risk, more and more companies are switching to packaged applications software. It is at this stage in the SDLC that the "make or buy" decision must be taken. We have analysed our user's requirements and can use these as selection criteria in searching for an appropriate package to purchase and install. Where there is no suitable package available we can still look to other innovative ways of obtaining the software, such as hiring contract staff or appointing a software house to build the system for us. Purchasing pre-written software will obviously mean the detailed systems design, coding and testing phases of the project are bypassed, depending on the need for customisation of the final system. In later courses we will look at the package selection process in more detail.

The objective of the **design stage** is to determine exactly how the new system will work, and to communicate this information in a document referred to as the detailed systems specification. If we take the analogy of an architect building a house, in the analysis stage he has determined the feasibility of the project and identified the owner's requirements in terms of the positioning of the house on the plot, size and architectural style, number of rooms and so on. The architect may even have built a small model to demonstrate the look and feel of the new dwelling. Once the owner is happy that the proposed house meets his requirements, the architect must communicate the detailed design to the builders. This would entail the drawing of a detailed plan of the house, specifying exactly how every part of the building is constructed and defining dimensions, materials, construction techniques etc.

We need to go through the same process when designing computer systems. This includes the design of:

- the technical platform on which the software will run. The new application may need new hardware, operating systems software and network connections
- output reports and enquiry screens
- input forms and data capture procedures
- physical file and database layouts
- description of the workings of each program module
- new clerical processes and procedures that will be required to interface with the new system.

Whereas in the analysis stage the emphasis was on identifying the user's needs with little concern for the way the computer would be used, the design stage also requires user involvement for the approval of detailed design, but the physical constraints imposed by the computer are also of major importance. Gane and Sarson [1979] define the objectives of structured design as follows:

"The most important objective of design, of course, is to deliver the functions required by the user. If the logical model calls for the production of pay cheques and the design does not produce pay cheques, or does not produce them correctly, then the design is a failure. But given that many correct designs are possible, there are three main objectives which the designer has to bear in mind while evolving and evaluating a design:

- **Performance**. How fast the design will be able to do the user's work given a particular hardware resource.
- **Control**. The extent to which the design is secure against human errors, machine malfunction, or deliberate mischief.
- **Changeability**. The ease with which the design allows the system to be changed to, for example, meet the user's needs to have different transaction types processed.

The output from systems design is a detailed design specification incorporating technical, input, output, data and process specifications. In the past, much of the information was communicated in written form which was difficult to understand and often ambiguous. Imagine the builder having to construct a house from a written description. Like the output from analysis we have a number of innovative tools to help users and developers understand and communicate the workings of the system. These include data models and data dictionaries, screen and report layouts, structure charts and pseudo-code. We will look at most of these later in this chapter.

12.1.4 Systems Build

In this stage we program the new system. If the system has been purchased "off-the shelf", this phase would consist of the customisation of the system. The success of the implementation stage is heavily reliant on the previous stages. If the analysis stage was poorly enacted, the system may not be what the user requires. Poor design will make it difficult for the programmer to construct the system, and it may be inefficient and difficult to maintain.

However, if the required effort and expertise is invested in analysis and design, there will be a precise specification of what to build available to the IS programmers and technical staff.

Unlike the previous stages, the programming stage can be undertaken as a number of separate tasks, performed in parallel. Programmers can code, data base administrators set up the database, hardware suppliers install and test networks and equipment, and we can begin to train the end-users to prepare them for the implementation phase. With so much happening, and with the need for some tasks to be completed before others begin, the analyst must develop a detailed project implementation plan to ensure tasks are scheduled and delays are quickly identified and addressed. Programming includes the following steps:

- database construction
- program coding and testing

- systems testing to check the system can handle expected loads and meets physical performance criteria
- finalise manual procedures

12.1.5 Systems Implementation

This entails the transition of the completed system into the operational environment, and includes the following tasks (some of which will already have been started in earlier phases):

- installation and testing of any new hardware, systems software and network infrastructure
- train users and operations staff
- transfer data (data conversion)from the manual or old system to the new database where necessary
- perform acceptance testing. This requires careful planning to ensure all data flows, error procedures, interfaces, controls and manual procedures are tested
- complete project documentation

The change over carries some risk, as failure of the new system may result in the organisation being unable to do business, There are a number of approaches to converting from the old system to the new. The least risky is to run the new system in parallel with the old until the new system is stable. There is obviously a cost to running both systems. Another approach is to convert one part of the organisation at a time (for example one branch office at a time). This method (known as the pilot method) reduces risk and allows the development team to focus on one area. This approach can cause some integration problems as part of the organisation is running on the old system and part on the new. A similar approach is the phased implementation method where organisations convert to a large system one step at a time. For example they may start with the stock system, then implement debtors and finally the order entry system. Some organisations use the **big bang** approach and just switch over from the old to the new system. This option is obviously high risk as there is no system to fall back on in an emergency.

When the new system has been in operation for a few months, a **post-implementation audit** should be carried out. This audit must ascertain whether the project has achieved the initial objectives specified in terms of:

- meeting initial scope and objectives
- anticipated costs and benefits to the organisation
- user satisfaction with the new system
- performance (response/turnaround time, reliability)
- adherence to standards
- quality of final product
- project performance in terms of cost and time.

This exercise will help to highlight problems that require maintenance of the new system and provide valuable feedback to IS management and project teams to assist in future development exercises.

12.1.6 Maintenance

Finally resources will be required to maintain and enhance the system over its operational life which can vary between 4 and 10 years. There is normally a formal hand-over of the system from the project team to the maintenance team. This will ensure that there is a defined time when the project is completed and that all the required documentation is produced. There are many systems in existence that are still supported by the original developer; and all knowledge of the system exists only in that individual's head. The problem is that when this person leaves (or worse gets run over by a bus), there is no one with any knowledge of the system and the organisation is at risk.

Research has shown that this is the most expensive stage of the life cycle as **program bugs** (as a result of poor design or bad coding and testing) or **enhancements** (poor analysis of user's requirements or changes to the business) require continual analysis and programming effort.

The following table summarises the important tasks in the six stages of the SDLC and highlights the main deliverables from each task.

Stage	Tasks	Deliverables
Preliminary Investigation	Problem Definition Scope and Objectives Data Gathering Risk Assessment Feasibility Analysis	Project Charter Feasibility Study
Systems Analysis	Data Gathering Systems Modelling User Requirements Definition	User Requirements Specification
Systems Design	Make or Buy Decision Physical Systems Design Technical Design	Detailed Systems Specification
Systems Build	Programming and testing Platform Implementation	Production System
Systems Implementation	User Training Data Conversion Systems Conversion Post-Implementation Review	Live System
Systems Maintenance	Fix system "bugs" System enhancement	Working System

Figure 12-2: SDLC Tasks

12.2 Development of Structured Methodologies

New and innovative systems development techniques are frequently proposed by researchers and practitioners and, over time, these formal methods have replaced the traditional pragmatic approach to developing computer systems.

12.2.1 Structured Programming.

In the 1960's, the major concern in IS development environments was the efficient utilisation of expensive computer hardware. Programs were written in low level languages with little or no support documentation and, over time, the code became almost impossible for maintenance programmers to understand and fix. So arose the need to introduce a set of standard rules and procedures for writing programs, often referred to as structured programming. Some of the key techniques in the structured programming approach include:

- a limited set of simple control structures (to control branching and looping)
- standard naming conventions for data and procedures
- self documenting programs.

Structured programming techniques ensured that programs were easier to write and test and much easier to maintain.

12.2.2 Structured Design.

In the mid 1970s the focus in systems development moved from program coding to systems design. Computer applications were becoming more complex with the introduction of large on-line, integrated systems.

IS researchers, and in particular Larry Constantine, studied the problems of program size and complexity, and determined that, as a problem grew in size, so there was a more than proportional growth in the complexity of the problem and therefore in programming time. He advocated that all systems should be made up of small modules each no longer than fifty lines of program code.

Fragmenting a problem into a number of modules can be done in many ways; and he urged that the best technique would be to segment the program by function (task) with each module being as functionally independent of other modules as possible. This would ensure that changes to one program module were unlikely to affect other modules.

This technique was known as **structured design;** and a graphical representation of the modules and their relationships known as a structure chart, was developed to assist in the process.

12.2.3 Structured Analysis.

In the late 1970s the focus in systems development moved again, this time to the analysis stage. IS professionals had formalised the design and coding of computer software but had neglected the most important development issue – what are the user's real requirements. Written specifications were the main source of communication throughout the project. In the same way that architects would never attempt to describe a building in a letter, so analysts needed tools and techniques which could be used to define and communicate the user's requirements to the systems design stage. These structured tools and techniques included:

- **Data Flow Diagrams** (DFD). These diagrams are used to show how data flows between the various processes in the system. DFD's are an excellent communication tool as they are simple enough for users to understand and yet detailed enough to form the basis for the systems design process. A number of DFD techniques has been developed since the original work was published by Tom De Marco in 1978. However, they all basically perform the same task. Data flow diagrams are one of the most used and popular IS charting techniques.

- **Entity Relationship Diagrams** (ERD). Entity relationship diagrams identify the major objects about which data is stored and chart their interrelationship. Like most formal techniques, its major value is that it forces the analyst into a structured and detailed investigation of all the data used in the system.

- **Decision Trees and Pseudo-code.** These tools enable the analyst to express process logic clearly and unambiguously. In the detailed analysis of an information system, the analyst often has to describe a logical process that the future system will have to perform. Examples of these could be the way that a personnel system calculates pension benefits for employees or a sales system calculates sales commissions. Decision trees are diagrammatic representations of the process logic, showing the conditions that need to be tested and the resulting activities in a simple tree-like structure. Pseudo-code can be described as "structured English". It permits the analyst to define process logic using English language couched in the logical structures inherent in program code. In reality it eliminates the verbosity and ambiguity from the English narrative.

- **Project Dictionary**. This tool enables the analyst to capture and catalogue the entire system specification on computer with the obvious advantages in reporting, cross-referencing and updating. In a database environment, data is no longer the property of each individual application but managed centrally as a corporate resource. Vast amounts of information about this data needs to be maintained, for example field names, types and lengths, validation rules, data structures and relationships. As systems move from the development to production environment, this data about data is transferred from the project dictionary into a production data dictionary to enable the database administrator (DBA) to build and maintain the corporate database.

As we will discuss later, most of the new computer assisted software engineering (CASE) tools are now built up round a project dictionary.

12.3 Alternative approaches to developing systems

Over the past 40 years, efforts have been made to improve the quality of new systems and to reduce the time and effort expended in their development. The following section provides an overview of some of the significant tools and techniques developed for this purpose.

12.3.1 Prototyping

One technique that has been incorporated successfully into the SDLC is prototyping. As the name suggests a **prototype** is a mock-up or model of a system for review purposes.

Looking at the traditional SDLC, one of the major problems is that the user is asked to provide detailed requirements prior to the system being built. Once a system is implemented he may find flaws in his original requirements or may see the possibilities of a new and improved approach.

For applications such as general ledger and payroll, the requirements are normally well understood (and usually standardised enough to suggest the use of packages) but many other areas such as personnel are unstructured and would benefit from the prototyping stage.

The two main approaches to the use of prototypes in the SDLC are:

- **discovery prototyping** where the analyst builds a skeleton of the final product in the analysis stage of the project in order that the user may better understand the workings of the final system. This prototype is normally built with a fourth generation language and while it is likely to include mock-ups of screens and reports, it is seldom a fully working model. The building and refining of the prototype is an iterative process between analyst and user and stimulates discussion on the functionality of the final product. Once the analyst and user are happy that the system's requirements have been identified, detailed requirement specifications are developed and the prototype is no longer required. In some instances the prototype may serve as the specification.

- **evolutionary prototyping** where a working model is built with the intention of later refining it into the operational system. While this technique would appear to have great advantages in terms of productivity, the original prototype is often thrown together and not properly designed.

Prototyping can offer IS developers many advantages in that it assists in clarifying user's requirements, improves user communication and commitment, should improve the functionality and quality of the user interface and will assist in identifying problems earlier in the development life cycle.

Where prototyping can be problematic is that it raises the user's expectations that systems are quick to build and changes are easy. In addition there is a lack of experienced prototypers and quality prototyping tools.

12.3.2 Joint Application Development (JAD)

One major problem in systems development projects is the lack of real communication, understanding and consensus between users, management and the development team. Instead of the traditional one on one interviews spread over weeks and often months, the JAD approach involves a series of highly structured workshops where stakeholders focus on any one of the planning, analysis, design and implementation stages of the life cycle. One obvious advantage of this approach is a reduction in the time it takes to develop systems. However the real benefits of JAD come from better user requirements through improved communication and conflict resolution. Successful JAD sessions often depend on the competence of the session leader (termed the facilitator), the scribe (who is responsible for documenting the output from the sessions), strong top management support and a mix of participants with expertise and responsibility for the area under discussion.

12.3.3 Computer Assisted Software Engineering (CASE)

There is a famous saying, “The cobblers children have no shoes.” and this is very relevant to IS. Here we have a classic example of a group of IS professionals, dedicated to computerising the organisation in which they work, while developing these computer systems via manual means.

CASE environments attempt to address this problem by offering a set of integrated electronic tools for use in the SDLC. During the first phase of system development, CASE products provide the analyst with computerised tools to complete and document the analysis and detailed design stages of the development project by graphically modelling the data requirements and the business process flows that the intended application has to address. These models attempt to give a visual representation of a part of the business operations, following one of many modelling standards. The resultant application model is then used as a blueprint for the actual implementation in computer code. The tools that are geared specifically for this modelling phase are referred to as *upper-CASE or U-CASE* tools.

Lower-CASE tools specialise in the second phase of system development: the actual generation of executable applications or advanced prototypes. This is typically achieved through the use of a generic application generator, although CASE tools tend to be independent of any specific database management system.

Integrated or *I-CASE* aims to automate both phases i.e. a combination of upper and lower-CASE tools in one single package.
Most CASE environments use an electronic project dictionary as a repository and can include:

- graphic tools for charting diagrams such as DFD’s, ERD’s and Structure charts
- 4th generation languages or application generators to assist with prototyping
- data dictionary facilities to record and maintain data about data, processes and other system details
- quality control facilities to check specifications and code for correctness
- code generators to reduce programming effort
- spreadsheet models to assist with cost/benefit analysis
- project management tools to plan, monitor and control the development cycle.

When CASE environments were originally developed in the mid 1980s, IS managers viewed them as a possible “silver bullet” to resolve the growing demand for computer systems. The promise of CASE was better quality systems, reduced development time, enforced standards and improved documentation.

As yet CASE tools have failed to make a major impact. CASE environments are complex, the cost of implementing CASE is high (both in terms of the CASE software and analyst training) and many organisations are looking to other solutions (packages and object orientation) to resolve their applications backlog.

12.3.4 Object Oriented Development (OOD)

Using the traditional development approach where the analyst designs procedures focused on the user's requirements can result in systems that are costly to develop and inflexible in nature. The **object oriented approach** attempts to build components (objects) that model the behaviour of persons, places, things or concepts that exist in the real world and then to build systems from these components. We do not design and build unique systems for motor cars or televisions; they are mostly built up from a set of common, interchangeable components. Even when components are unique they are very similar to other components. In the same way we can construct computer systems from building blocks reusing objects from other systems, making modifications to similar objects or obtaining objects from commercial component libraries.

The OOD paradigm is only now gaining momentum in the market place and most programming languages and methodologies do not support OO development. One exception is Small Talk, the language credited with pioneering the OOP (object oriented programming) concept. Today many of the popular programming languages are appearing with OO versions (for example Pascal, C++, Visual Basic and even COBOL).

12.3.5 Other development tools

The above-mentioned new programming approaches were not the only attempts to improve developer productivity. The following presents some other development tools and approaches. Note that, since distinctions between the categories cannot always be perfect, some tools could be classified in more than one category.

§ *Visual programming tools.* The power of graphical user interfaces and object-orientation has spawned a number of high-level *front-ends* or *shells* to enable non-programmers to generate their own straightforward applications. These *visual programming tools* allow for the construction of applications by selecting, connecting, copying and arranging programming objects. For simple applications, there is no need for any code to be written at all since all required objects can be copied from a large library with all commonly used, pre-configured objects and their associated standard methods.

§ *Report generators* are generally associated with database management systems and allow users to create ad-hoc, customised reports using the data in the database by specifying the various selection criteria and the desired report layout.

§ *Application generators* consist of standard building blocks that can be combined or customised to create the required systems. The user specifies the inputs, the output requirements and the various data validations and transformations. Screen and report *painters* allow on-line, visual layout of input and output modules. Generally, these generators are supported by a comprehensive database management system that integrates the data dictionary, graphics and reporting modules as well as other utilities such as data and process modelling, security facilities, decision support modules and *query-by-example (QBE)* languages.

§ *Logic programming for knowledge based systems.* Programmers quickly discovered that conventional programming languages were inadequate to develop advanced *knowledge-based* applications, such as expert systems and other artificial intelligence systems. These systems require reasoning capabilities and have knowledge representation requirements that are difficult to implement using procedural languages. This led to the development of *logic programming* languages such as *LISP* and *Prolog*. The use of these languages is generally confined to researchers and scientists. Today, a number of shells has been developed that allow the automatic generation of straight-forward knowledge-based systems.

§ *End-user applications.* Today's end-user productivity applications have extensive programming capabilities and allow for customisation by means of *macros* (pre-recorded sequences of commands, keystrokes) and *formulae*. A spreadsheet is essentially a model developed by an end-user whereby the equations (or data transformations) have been programmed by means of formulae. Many of these formulae look similar to the statements in programming languages. The following statement would calculate someone's weekly wages taking into account an overtime rate of 50%, using Microsoft Excel or Access.

```
= IF ( hours > 42 , hours * wage , 42 * wage + ( 1.5 * wage ) * ( hours – 42 ) )
```

12.4 Critical success factors

Regardless of the development approach that may be used, a number of factors have been identified that are critical to ensuring the success of a systems development project:

- well defined system objectives
- careful test of feasibility
- top management support
- user involvement to ensure strong commitment
- rigorous analysis to ensure detailed, unambiguous user requirements
- sound detailed design to ensure an efficient, quality, maintainable system
- project management to ensure the development team is managed and controlled.

12.5 South African Perspective

Research by a group of Scandinavian computer scientists has suggested that prototyping is better suited than the traditional SDLC to systems development projects undertaken in developing countries. Although the SDLC provides a rigorous development methodology intended to generate clearly defined system requirements, it does not take into account problems resulting from social and cultural factors. These include the uncertain availability of technical skills, user anxiety about technology, and the need for adaptability of the final product to differing local conditions. A further dimension to this approach is the need to train users in basic computer literacy skills and to inform them about the business role of IS *before* any attempt is made to elicit system requirements. Once workers have been empowered in this way, they are able to provide more valuable participation in the development of a system. Developers must also see the project as a mutual learning experience, since they need to understand how future changes in the business environment may affect system requirements.

12.6 Beyond the Basics

Web pages can contain multimedia effects and interactive capabilities, and the development of web pages involves using a variety of special components. Among the jargon and acronyms that you may encounter are the following:

- *Hypertext Markup Language (HTML)* is not actually a programming language, but has specific rules for defining the formatting of text, graphics, video and audio on a web page. *Tags* are used to indicate how a page should be displayed on your screen, and the details underlying links to other web pages.
- Interactive elements such as scrolling messages, pop-up windows and animated graphics are controlled by small programs, generally *scripts*, *applets* or *ActiveX controls*. Basically, a script is an interpreted program that runs on the client computer, as opposed to an applet, which is a compiled program running on the client computer. ActiveX controls are object-oriented technologies that allow components on a network to communicate with one another.
- Information is sent and received between your computer and a web server via the *common gateway interface (CGI*), which is the communications standard that defines how a web server communicates with outside sources such as a database. CGI programs are frequently written using scripting languages such as JavaScript, which is simpler to use than the full Java language.
- Web pages created using *dynamic HTML* can automatically update their content on the client's machine, without having to access the web server, making them more responsive to user interaction. *Extensible HTML* (XHTML) uses XML technology to display web pages on different types of display devices, while *wireless markup language* (WML) supports browsing on PDAs and cellular telephones. Finally *wireless application protocol* (WAP) specifies how wireless devices such as cellular telephones communicate with the Web.

12.7 Exercises

12.7.1 Stages of the SDLC

Read the following article, and then briefly explain, for each stage of the SDLC, which of the standard activities appear to have been omitted or not completed when developing the system, and what effect this had on the quality of the final product.

From: Machlis, S. "U.S. Agency Puts $71m System on Ice", Computerworld, 12 May 1997.

The U.S Agency for International Development (AID) last week confirmed that it suspended overseas use of a new computer system plagued by integration snafus, data transmission bottlenecks, and response times so slow that employee efficiency suffered. For now, 39 field sites will go back to using the agency's old system for core accounting services and procurement contracts while problems with the Washington-based computers are ironed out. "We need to get the core functionality established", said Richard McCall, AID's chief of staff.

The New Management System (NMS), budgeted at $71 million, has been under fire since it was deployed in October of 1997. The AID inspector general's office criticized NMS for data errors and slow performance. In some cases, users had to spend days trying to process a single transaction.

The new system can't handle the large amount of data that passes among AID offices, McCall said. The agency must decide whether to boost expensive satellite network bandwidth to handle real time transactions or move to some batch processing. "I don't think people understood the amount of data that would be transmitted over the system", he said.

Designers also initially failed to gasp the difficulty of integrating legacy accounting systems. "We thought we had three primary accounting systems," McCall said. But numerous infield alterations to basic systems over the years meant the agency had closer to 80 different accounting systems. Some of the resulting data didn't import correctly into the new system.

In addition, McCall said, system designers should have stayed focused on core requirements instead of trying to immediately add features that users requested after early tests. For example, some overseas employees wanted to be able to call up data from any foreign site. Although that is an attractive feature, he said, "that taxes the system. You don't really need that now."

12.7.2 Systems conversion

Four different methods have been described for the conversion to a new system: the parallel, pilot, phased or "big bang" approach. Compare these four alternatives in terms of the likely time frame involved, level of risk incurred, and user "buy-in" to the new system.